VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Multiplatformní aplikace pro vizualizaci a simulaci procesních Petriho sítí

Multi-platform Application for Process Petri Nets Visualization and Simulation

Student: Bc. Martin Čechák

Vedoucí diplomové práce: doc. RNDr. Ivo Martiník, Ph.D.

Ostrava 2017

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Martin Čechák**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T025 Systémové inženýrství a informatika

Téma: Multiplatformní aplikace pro vizualizaci a simulaci procesních Petriho sítí
Multi-platform Application for Process Petri Nets Visualization and Simulation

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teorie procesních Petriho sítí
3. Analýza dostupných aplikací pro simulaci Petriho sítí a využité technologie
4. Návrh a implementace aplikace pro vizualizaci a simulaci procesních Petriho sítí
5. Validace vyvíjené aplikace v praxi
6. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

REISIG, Wolfgang. *Understanding Petri Nets: modeling techniques, analysis methods, case studies.* Berlin: Springer, 2013. ISBN 978-3-642-33277-7.
HUANG, H., JIAO, L., T. CHEUNG and W. MAK. *Property-Preserving Petri Net Process Algebra In Software Engineering.* Singapore: World Scientific Publishing, 2012. ISBN 978-981-4324-28-1.
SHARAN, Kishori. *Learn JavaFX 8: Building User Experience and Interfaces with Java 8.* New York: Apress, 2015. ISBN 978-1-484211-43-4.
SMITH, Dave. *Android recipes: a problem-solution approach for Android 5.0.* Fourth ed., New York: Apress, 2015. ISBN 978-1-484204-76-4.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. RNDr. Ivo Martiník, Ph.D.**

Datum zadání: 18.11.2016

Datum odevzdání: 21.04.2017

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Zdeněk Zmeškal
děkan fakulty

Prohlášení

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.

V Karviné dne 19. 4. 2017

.....

Bc. Martin Čechák

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce doc. RNDr. Ivu Martiníkovi Ph.D. za pomoc, cenné rady z oblasti teorie Petriho sítí a inspiraci k výběru tématu.

Obsah

1 Úvod

Aktuálním tématem v oblasti informačních technologií jsou multiplatformní aplikace. Z důvodu vysokých nákladů tvorby jedné aplikace pro více platforem pomocí nativních technologií jsou využívány při vývoji nové postupy. Business aplikace, tedy klienti služeb poskytovaných servery (např. CRM systémy), jsou často vyvíjeny ve webovém prostředí, jehož uživatelské rozhraní je poté přizpůsobeno pro jednotlivá zařízení. Aplikace je však napsána jen jednou, a to například ve skriptovacím jazyku JavaScript s využitím frameworku React nebo Angular. Výpočetně náročnější aplikace s rozsáhlou funkcionalitou, jako jsou např. simulační programy, CAD systémy, programy pro 3D modelování nebo programy pro stříhání videa, jsou již většinou napsány v nativních programovacích jazycích, v programovacích jazycích používající virtuální stroj (Java, C#), popřípadě skriptovacích jazycích, jakým je např. Python. V těchto případech je však nutné přizpůsobit aplikaci podporovaným platformám.

Tato diplomová práce je zaměřena právě na druhou zmíněnou kategorii aplikací. V oblasti aplikací pro podporu návrhu, modelování a verifikaci Petriho sítí ke dni výběru tématu této práce chyběla aplikace, která by umožňovala práci současně na desktopových i mobilních zařízeních. Typicky jde v tomto případě například o možnost práce na počítači a tabletu. Jedním z dalších motivů vývoje uvedené multiplatformní aplikace je dále i možnost snadné implementace nových podtříd Petriho sítí.

Důležitou volbou byla technologie, která by umožňovala implementovat celou aplikaci, nebo alespoň její velkou část, pouze jednou, a poté mít možnost ji přizpůsobit, případně přeložit, pro jednotlivé platformy. Pro účely této práce byl vybrán programovací jazyk Java, framework JavaFX (viz kapitola 3.2.2) a pro překlad na mobilní platformy byl využit projekt Gluon (viz kapitola 3.2.3).

Nedílnou součástí vývoje této aplikace je samozřejmě podrobná znalost teorie Petriho sítí, a proto je teoretická část diplomové práce plně věnována právě jí.

Praktická část diplomové práce je uvedena analýzou vybraných a již dostupných aplikací pro modelování Petriho sítí, a to jak komerčních, tak i freewarových. Dále jsou zde ve stručnosti popsány využité technologie a cílové platformy určené pro vývoj a provoz v rámci této práce nově navržené

a implementované multiplatformní aplikace pro podporu návrhu, modelování a verifikaci Petriho sítí.

V rámci této práce pak byla nově vyvinutá aplikace validována v praxi na příkladu jednoduchého procesu z hlediska její funkcionality, výpovědní hodnoty, přehlednosti a možností analýzy vzniklých dat.

Cíle diplomové práce jsou následující:

- na základě uživatelského příběhu a teorie P/T Petriho sítí specifikovat požadavky na vyvíjenou aplikaci,
- analyzovat dostupný software v oblasti návrhu, simulace a verifikace Petriho sítí a verifikovat tím požadavky na nově vyvíjenou multiplatformní aplikaci,
- navrhnout a implementovat aplikaci pro návrh, simulaci a verifikaci sekvenčních P/T Petriho sítí s ohledem na její multiplatformní charakter,
- ověřit, v jakém rozsahu a s jakými omezeními je prostřednictvím projektu Gluon Mobile možné aplikaci implementovanou v programovacím jazyku Java ve frameworku JavaFX přeložit a zprovoznit v prostředí mobilních platforem s operačními systémy Google Android a Apple iOS,
- validovat vlastnosti nově vyvinuté multiplatformní aplikace provedením návrhu a verifikace vybraných procesů s využitím teorie procesních Petriho sítí.

Druhá kapitola této práce je věnována teorii Petriho sítí a zejména pak jejich třídy tzv. procesních Petriho sítí. Petriho sítě jsou zde studovány z hlediska jejich historie, využití v praxi, definice, vlastností a vybraných podtříd.

Ve třetí kapitole je provedena analýza dostupného softwaru pro podporu modelování a simulace Petriho sítí. V rámci této kapitoly jsou studovány i technologie, které byly využity při návrhu a implementaci vyvíjené multiplatformní aplikace a cílové platformy pro její implementaci.

Čtvrtá kapitola je věnována samotnému návrhu a implementaci aplikace určené k návrhu, simulaci a verifikaci vybrané třídy procesních Petriho sítí. V úvodu kapitoly je rozebrána metodika vývoje aplikace a pomocný software důležitý pro její vývoj. Následuje návrh jednotlivých modulů a představení samotné aplikace z hlediska jejího uživatelského rozhraní a funkcionalit.

V páté kapitole je realizováno ověření vlastností nově vyvinuté multiplatformní aplikace provedením návrhu a verifikace vybraných procesů s využitím teorie procesních Petriho sítí. Dále zde jsou uvedeny i návrhy pro její budoucí vývoj.

2 Teorie procesních Petriho sítí

Tato kapitola je věnována teorii procesních Petriho sítí. V jejím úvodu je stručně uvedena historie Petriho sítí, uvedena definice třídy Petriho sítí, její podrobnější členění na základní podtřídy a studovány jejich základní vlastnosti. Definici a vlastnostem třídy procesních Petriho sítí je pak věnován závěr kapitoly.

2.1 Historie Petriho sítí

Petriho sítě, stejně jako jejich grafická notace a metodika, vznikly už na konci 30. let 20. století, plány na jejich výraznější rozvoj však překazila druhá světová válka. Původně šlo o nástroj pro simulaci chemických procesů. (Reisig, 2013)

Až v 50. letech 20. století byly možnosti využití Petriho sítí rozšířeny o simulaci činnosti počítačů, které se v té době dynamicky rozvíjely Pojem Petriho sítě zavedl německý matematik Carl Adam Petri (12. 7. 1926 – 2. 7. 2010) ve své doktorské práci „Kommunikation mit Automaten“ v roce 1962. (Reisig, 2013)

Během posledních šedesáti let bylo učiněno mnohé pro pochopení počítačů a distribuovaných systémů jako takových. Jedním z matematických nástrojů, který současně umožňuje grafickou reprezentaci těchto systémů a jejich formální analýzu jsou právě Petriho sítě.

Nutnost přizpůsobit se jednotlivým doménám využití (viz následující kapitola) vedla k tomu, že bylo zavedeno velké množství tzv. tříd Petriho sítí, které svými vlastnostmi umožňují daný problém lépe modelovat. Vedle klasických černobílých P/T Petriho sítí zavedených v roce 1966 s omezenými modelovacími možnostmi se dnes využívají například stochastické, vysokoúrovňové, fuzzy, barevné nebo objektově-orientované Petriho sítě, které jsou zaměřený na modelování a verifikaci objektově orientovaných programových systémů. (Martiník, 2015)

2.2 Petriho sítě v praxi

Jedním z řešených problémů v oblasti informačních technologií je tvorba systémů obsahujících technologické komponenty, které následně kooperují v automatizovaném prostředí. Petriho sítě napomáhají k návrhu, simulaci a verifikaci těchto systémů, které jsou složeny z celé škály rozdílných komponent, jako např. klienti, výrobci, uživatelé, agenti, sdílené prostředky, atd. Petriho sítě jsou pak jedním z nejstarších nástrojů pro modelování všech typů paralelních i neparalelních

systémů a na rozdíl od ostatních metodik jsou používány již více než 50 let. Petriho sítěmi lze modelovat mnohé běžné problémy za předpokladu, že je návrhář převede do podoby systému.

Teorie Petriho sítí má své široké uplatnění i v oblasti ekonomických věd. Lze je s úspěchem aplikovat při modelování různých typů ekonomických systémů a popisovat prostřednictvím jejich vlastností řídicí a datové toky a informační závislosti uvnitř těchto systémů. Pomocí Petriho sítí lze tyto systémy navrhovat, analyzovat, simulovat a verifikovat.

Výsledky simulace je pak možno využit k optimalizaci systému. Pro tyto účely existují i alternativní metody, jakými jsou lineární, kvadratické nebo stochastické programování. Uvedené metody jsou však čistě matematické a chybí zde grafická reprezentace modelovaných systémů.

V následujících odstavcích jsou stručně popsány užší oblasti aplikace Petriho sítí.

Řízení výroby

V této oblasti se jedná zejména o modelování a verifikaci komplexních výrobních systémů a jejich komponent, produkčních linek, flexibilních výrobních systémů, automatizovaných montážních linek, produkčních systémů, výrobních systémů se společnými sdílenými zdroji, stavů uváznutí (tzv. deadlock) ve vybraných typech Just-in-Time výrobních systémů, apod. (Dicesare et al., 1993)

Dále pak jde o oblast úloh zaměřených na rozvrhování činností v průmyslových robotických systémech. (Shen et al., 1992).

Procesní analýza

V procesní analýze jde o business procesy a návrh a implementaci informačních a znalostních systémů pro jejich podporu. (Van Hee et al., 2002)

Z pohledu nákladů business procesů jde o analýzu inkonzistence, duplikace operací, vybraných aspektů implementace a následných možnosti jejich redukce. (Poliaschuk, 2011)

Dále lze v rámci uvedených procesů modelovat finanční toky a jejich charakteristiky. (Chen et al., 1992)

Projektové řízení

Jedním z důležitých aspektů projektového řízení jsou zdroje (lidské, hmotné, nehmotné), jejich řízení, řešení vytíženosti a optimalizace využití. (Jeetendra et al., 2000)

Logistika

Pomocí Petriho sítí lze v oblasti logistiky modelovat hlavně komplexní dopravní a transportní systémy, a tím umožnit jejich optimalizaci. (Aalst, 1992)

2.3 Základní matematické pojmy a symboly

V celém následujícím textu této práce jsou použity standardní symboly pro označení množin, čísel, relací, funkcí, logických operátorů, sekvencí, vektorů, matic a dalších matematických pojmů. Dále jsou použity standardní logické operátory ¬ (negace), ∧ (konjunkce), ∨ (disjunkce), ⇒ (implikace), ⇔ (ekvivalence), = (rovnost), ≠ (nerovnost) a kvantifikátory ∃ (existenční kvantifikátor) a ∀ (všeobecný kvantifikátor) v jejich obvyklém významu známém z predikátového počtu. (Martiník, 2015)

Množiny označujeme velkými tiskacími písmeny, např. A, B, C. Kardinalitu množiny A označujeme symbolem $|A|$. Množiny s konečnou kardinalitou lze explicitně zadat výčtem jejich prvků, který uvádíme ve složených závorkách, např. $A := \{1, 2, 3\}$. Prázdnou množinu, tj. množinu, která neobsahuje žádné prvky, označujeme standardním symbolem ∅. Pro každou entitu x a množinu S je výraz $x \in S$ (tj. x je prvkem množiny S) buďto pravdivým, nebo nepravdivým výrokem. Množinu všech přirozených čísel označujeme symbolem $\boldsymbol{N}$ (tj. $\boldsymbol{N} := \{1, 2, 3, \ldots\}$), množinu všech nezáporných celých čísel symbolem $\boldsymbol{N}_0$ (tj. $\boldsymbol{N}_0 := \{0, 1, 2, 3, \ldots\}$), množinu všech celých čísel symbolem $\boldsymbol{Z}$ (tj. $\boldsymbol{Z} := \{\ldots, -2, -1, 0, 1, 2, \ldots\}$). Množiny je možno zadávat i prostřednictvím speciální notace jako kolekce entit, které vyhovují danému logickému predikátu. Je-li např. x proměnná vybraného datového typu a $P(x)$ logickým predikátem o tomto typu dat, lze danou množinu A specifikovat prostřednictvím výrazu $A := \{x \mid P(x)\}$ reprezentujícího množinu všech entit daného datového typu, pro které je logický predikát $P(x)$ pravdivý. Nechť A a B jsou množiny. Výrazem $A \subseteq B$ vyjadřujeme skutečnost, že množina A je podmnožinou množiny B včetně možnosti, že $A = B$. Výrazem $A \subset B$ pak vyjadřujeme skutečnost, že množina A je vlastní podmnožinou množiny B (tj. $A \subseteq B$ a $A \neq B$). Výrazem $A \cap B$ označujeme průnik množin

A a B, tj. $A \cap B := \{x \mid (x \in A) \wedge (x \in B)\}$. Výrazem $A \cup B$ označujeme sjednocení množin A a B, tj. $A \cup B := \{x \mid (x \in A) \vee (x \in B)\}$. Výrazem $A \setminus B$ označujeme rozdíl množin A a B, tj. $A \setminus B := \{x \mid (x \in A) \wedge (x \notin B)\}$. Je-li A množina s konečnou kardinalitou, budeme symbolem $\mathscr{P}(A)$ označovat množinu všech podmnožin množiny A. (Martiník, 2015)

Nechť S a T jsou množiny. **Kartézským součinem** $S \times T$ rozumíme množinu všech uspořádaných dvojic (s, t), v nichž s je prvkem množiny S a t je prvkem množiny T, tj. $S \times T := \{(s, t) \mid (s \in S) \wedge (t \in T)\}$. Obecně kartézský součin $S_1 \times S_2 \times \ldots \times S_n := \{(s_1, s_2, \ldots, s_n) \mid (s_1 \in S_1) \wedge (s_2 \in S_2) \wedge \ldots \wedge (s_n \in S_n) \wedge (n \in \mathbf{N})\}$. (Martiník, 2015)

Funkcí f rozumíme matematickou entitu se kterou je jednoznačně asociován její **definiční obor** (označujeme jej symbolem $\boldsymbol{dom}(f)$) a **obor hodnot** (označujeme jej symbolem $\boldsymbol{cod}(f)$), které musí být množinami. Pro každý prvek x definičního oboru funkce f má funkce f přiřazenu svou **funkční hodnotu**, kterou označujeme symbolem $f(x)$ a která je prvkem oboru hodnot funkce f. Výrazem $f: S \rightarrow T$ vyjadřujeme skutečnost, že definičním oborem funkce f je množina S a oborem hodnot této funkce je množina T. (Martiník, 2015)

2.4 Základní členění Petriho sítí

Základní členění Petriho sítí je na nízkoúrovňové Petriho sítě, jejichž základním charakteristickým znakem je nerozlišitelnost tzv. značek popisujících stav modelovaného systému, a vysokoúrovňové Petriho sítě, které již využívají vzájemně rozlišitelné značky. Obě kategorie se pak člení do jednotlivých podtříd Petriho sítí. V této práci se prvotně věnujeme nízkoúrovňovým P/T Petriho sítím a její podtřídě procesním P/T Petriho sítím.

V následující kapitole jsou uvedeny základní charakteristiky Petriho sítí – tj. pojem grafu, bipartitního grafu, sítě, místa, přechodu, hrany a značky.

2.5 Definice P/T Petriho sítí

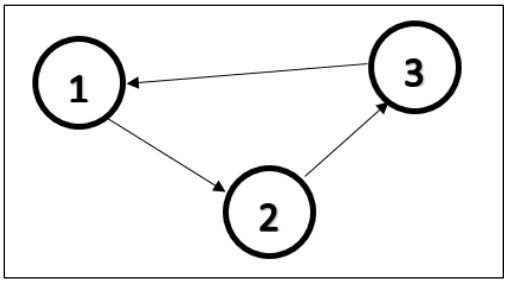
P/T Petriho sítě jsou bipartitní orientované grafy, tj. grafy se dvěma kategoriemi uzlů, které jsou mezi sebou spojeny hranami. V rámci této kapitoly budeme definovat pojem P/T Petriho sítě na základě definice grafu, bipartitního grafu, sítě a Petriho sítě.

2.5.1 Orientovaný graf

Orientovaným grafem rozumíme uspořádanou dvojici $OG := (U, H)$, přičemž platí:

- U je konečná neprázdná množina **uzlů**,
- $H \subseteq U \times U$ je konečná množin **hran**.

Příklad orientovaného grafu je uveden na obrázku 2-1. Uzly, které jsou reprezentovány kružnicemi a ohodnoceny čísly, patří do množiny U.



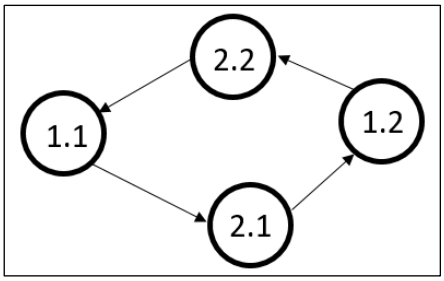
Obrázek 2-1:Orientovaný graf, zdroj: vlastní

2.5.2 Bipartitní orientovaný graf

Bipartitním orientovaným grafem rozumíme uspořádanou trojici $BOG := (U_1, U_2, H)$, přičemž platí:

- U_1 je konečná neprázdná množina **uzlů**,
- U_2 je konečná neprázdná množina **uzlů** a platí: $U_1 \cap U_2 = \varnothing$,
- $H \subseteq (U_1 \times U_2) \cup (U_2 \times U_1)$ je konečná množin **hran**. (Martiník, 2015)

Příklad bipartitního orientovaného grafu je uveden na obrázku 2-2. Uzly jsou zde reprezentovány kružnicemi a náleží množinám U_1 a U_2 (dle prvního čísla před tečkou).



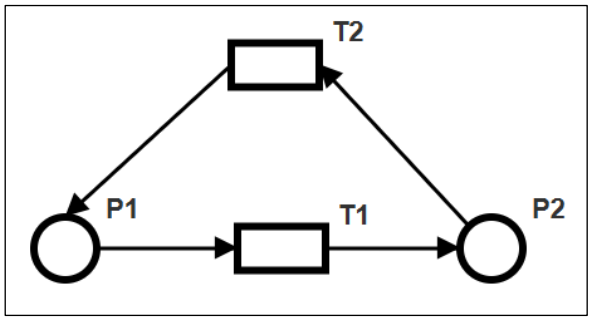
Obrázek 2-2:Bipartitní orientovaný graf, zdroj: vlastní

2.5.3 Síť

Sítí (net) rozumíme uspořádanou trojici *NET* := (*P*, *T*, *A*), přičemž platí:

- *P* je konečná neprázdná množina míst,
- *T* je konečná množina přechodů, $P \cap T = \emptyset$,
- *A* je konečná množina hran, $A \subseteq (P \times T) \cup (T \times P)$. (Martiník, 2015)

Příklad sítě je uveden na obrázku 2-3. Místa P1 a P2 z množiny P jsou zde reprezentovány kružnicemi, přechody T1 a T2 z množiny T jsou pak reprezentovány obdélníky.



Obrázek 2-3: Síť, zdroj: vlastní

Nechť *NET* := (*P*, *T*, *A*) je síť. Pak definujeme následující funkce:

InputNodes: $(P \cup T) \rightarrow \mathscr{P}(P \cup T)$ tak, že:

$\forall x \in (P \cup T)$: *InputNodes*(x) := $\{y \in (P \cup T) \mid \exists a \in A: a = (y, x)\}$.

Množinu *InputNodes*(*x*) budeme zkráceně označovat symbolem $\bullet x$.

OutputNodes: $(P \cup T) \rightarrow \mathscr{P}(P \cup T)$ tak, že:

$\forall x \in (P \cup T)$: *OutputNodes*(x) := $\{y \in (P \cup T) \mid \exists a \in A: a = (x, y)\}$.

Množinu *OutputNodes*(*x*) budeme zkráceně označovat symbolem $x\bullet$. (Martiník, 2015)

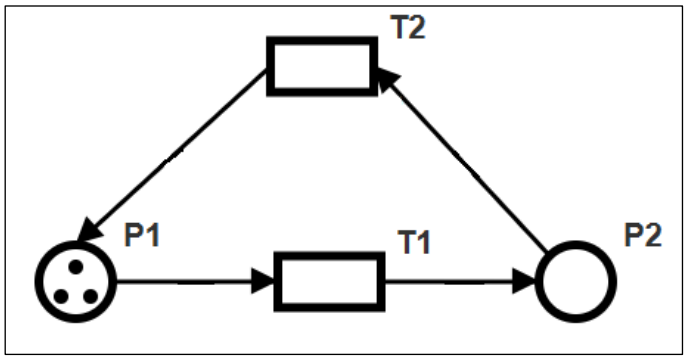
2.5.4 Petriho síť

Petriho síť je tedy sítí obsahující ve svých místech tzv. **počáteční značení** M_0 **sítě**, tj. s každým místem sítě je obecně asociováno nezáporné celé číslo vyjadřující počet anonymních značek nacházejících se v tomto počátečním značení M_0 v příslušném místě. Počátečním **značením** Petriho sítě tedy rozumíme funkci M_0: $P \rightarrow \boldsymbol{N}_0$ vyjadřující počáteční stav modelovaného systému. (Martiník, 2015)

Petriho síť tedy lze definovat jako uspořádanou dvojici *PN* := (*NET, M*$_0$) kde:

- *NET* := (*P*, *T*, *A*) je síť,
- *M*$_0$ je počátečním značením sítě, $M_0{:}P \to \boldsymbol{N}_0$. (Martiník, 2015)

Na obrázku 2-4 je zobrazena Petriho síť ve svém počátečním značení *M*$_0$. Anonymní značky náležící místu *P1* jsou reprezentovány tečkami.



Obrázek 2-4:Petriho síť, zdroj: vlastní

2.5.5 P/T Petriho síť

P/T Petriho síť rozšiřuje zavedený pojem Petriho sítě o tzv. hranovou funkci *AF*. **P/T Petriho sítí** tedy rozumíme uspořádanou trojici *PTN* := (*NET*, *AF, M*$_0$) kde:

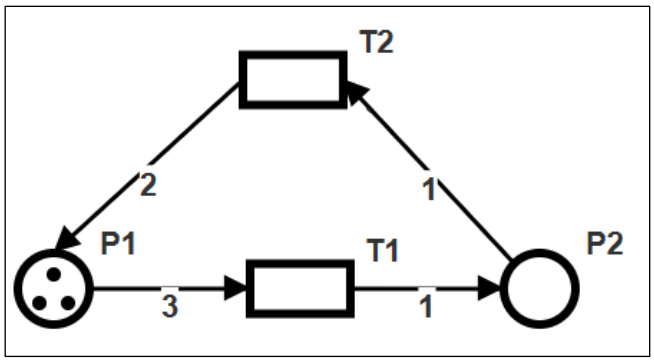
- *NET* := (*P*, *T*, *A*) je síť,
- *AF*: $(P \times T) \cup (T \times P) \to \boldsymbol{N}_0$ je **hranová funkce,**

  $AF(x, y) > 0 \Leftrightarrow (x, y) \in A$, $AF(x, y) = 0 \Leftrightarrow (x, y) \notin A$, kde $x, y \in P \cup T$,
- *M*$_0$ je počátečním značením sítě, $M_0{:}\ P \to \boldsymbol{N}_0$.

  (Huang, 2012), (Martiník, 2015)

P/T Petriho sítě budeme standardně zapisovat formou uspořádané pětice *PTN* := (*P*, *T*, *A, AF, M*$_0$).

Pro účely demonstrace hranové funkce *AF* je na obrázku 2-5 znázorněna P/T Petriho síť ve svém počátečním značení *M*$_0$ a odlišnými hodnotami hranové funkce *AF* asociovanými s jejími jednotlivými hranami (viz číslo uprostřed každé z hran P/T Petriho sítě).



Obrázek 2-5: P/T Petriho síť, zdroj: vlastní

2.6 Dynamika Petriho sítí

Všechny základní komponenty a pojmy nutné pro pochopení principu dynamiky P/T Petriho sítí byly definovány v předchozích kapitolách, nyní je tedy možné formálně tento pojem zavést.

2.6.1 Koncepce dynamiky

Dynamikou P/T Petriho sítí se rozumí simulace jejího vývoje stavů (značení), které jsou realizovány tzv. prováděním přechodů.

Základním principem simulace dynamických vlastností modelovaného systému je tzv. realizace provádění přechodů, které převádí P/T Petriho síť mezi jejími tzv. značením (viz kapitola 2.4.4).

Značením M P/T Petriho sítě $PTN := (P, T, A, AF, M_0)$ rozumíme funkci $M: P \rightarrow \mathbf{N}_0$, která každému jejímu místu $p \in P$ přiřazuje jistý konečný počet anonymních značek. Změna značení M dané P/T Petriho sítě provedením přechodu $t \in T$ je možná právě tehdy, jestliže všechna místa $p \in P$ z množiny $\bullet t$ obsahují minimálně takový počet značek, který je dán hodnotou funkce $AF(p, t)$. Přechod t je v tomto značení M sítě PTN proveditelný, Přechod t je v tomto značení M sítě PTN proveditelný, tuto skutečnost budeme dále označovat výrazem t ***en*** M, a může být proveden, a může být proveden.

Provedení (realizace) přechodu t je atomická operace, v jejímž rámci je:

- ze všech míst $p \in P$ z množiny $\bullet t$ odebráno právě $AF(p, t)$ značek,
- do všech míst $p \in P$ z množiny $t \bullet$ je přidáno právě $AF(t, p)$ značek.

Provádění přechodů sítě PTN je přitom možné realizovat v sekvenčním nebo distribuovaném režimu. Na první zmiňovaný režim je v rámci této práce kladen vyšší důraz.

2.6.2 Sekvenční běh

Sekvenční běh lze chápat jako posloupnost jednotlivých značení P/T Petriho sítě PTN, respektive průchod stavovým prostorem, jehož stavy jsou jednotlivá značení *M* sítě *PTN* a operátory reprezentují realizace náhodně vybraného proveditelného přechodu *t* z množiny přechodů *T*.

2.6.3 Distribuovaný běh

Zatímco sekvenční běh P/T Petriho sítě lze vyjádřit jako posloupnost jejich jednotlivých značení, reprezentací distribuovaného běhu je tzv. *kauzální síť*, která obsahuje všechny akce. Pro více informací viz Reisig (2013).

2.7 Analýza Petriho sítí

V této kapitole jsou definovány nejdůležitější pojmy z oblasti analýzy Petriho síti, mezi které patří stavové vlastnosti, zámky, pasti, P-invarianty a T-invarianty.

2.7.1 Stavové vlastnosti

Stavovými vlastnostmi jsou myšleny vlastnosti, které přetrvávají po celou dobu běhu systému reprezentovaného danou P/T Petriho sítí. Stavové vlastnosti se týkají jednoho nebo více míst studované P/T Petriho sítě, přičemž je studován počet značek nacházejících se v těchto místech v jednotlivých jejich značeních *M* dosažitelných z jejího počátečního značení M_0. Pomocí analýzy všech dosažitelných značení lze stanovit rovnice, nerovnice, popřípadě identity, platné pro danou P/T Petriho síť, kterým jsou věnovány následující podkapitoly. (Reisig, 2013)

Incidenční matice

Obecný orientovaný graf *OG* := (*U*, *H*) lze reprezentovat pomocí tzv. incidenční matice (viz výpis 2-1), která je obecně čtvercovou maticí. Je-li *OG* := (*U*, *H*) orientovaný graf, je počet řádků a sloupců této incidenční matice dán kardinalitou množiny *U*. S každým řádkem a sloupcem je pak asociován jeden z uzlů množiny *U*. Prvky dané incidenční matice pak nabývají následujících hodnot:

- **hodnoty 0** – pokud neexistuje orientovaná hrana, která propojuje uzel asociovaný s řádkem matice a uzel asociovaný se sloupcem matice,
- **hodnoty 1** – pokud existuje orientovaná hrana, která vede z uzlu asociovaného s řádkem matice do uzlu asociovaného se sloupcem matice.

Výpis 2-1: Incidenční matice orientovaného grafu z obr. 2-1, zdroj: vlastní

	u_1	u_2	u_3
u_1	0	1	0
u_2	0	0	1
u_3	1	0	0

V následujícím textu pak bude definována incidenční matice P/T Petriho sítě.

Vektorová reprezentace běhu P/T Petriho sítě

Nejdříve je nutné definovat tzv. vektorovou reprezentaci provádění přechodů P/T Petriho sítě. Je-li M_0 počáteční značení dané P/T Petriho sítě a t_1 je proveditelným přechodem v tomto počátečním značení M_0, vyjadřujeme výrazem:

$$M_0 \xrightarrow{t1} M_1 \qquad \textit{Vzorec 2-1}$$

Skutečnost, že po provedení přechodu t_1 byla daná P/T Petriho síť převedena do nového značení M_1. Přechod mezi uvedenými značeními sítě lze vyjádřit pomocí součtu vektorů:

$$M_1 = M_0 + t_1 \qquad \textit{Vzorec 2-2}$$

kde M_1 a M_0 jsou vertikálními vektory značení, jejichž rozměr je roven počtu míst obsažených v množině míst P dané P/T Petriho sítě a jejichž jednotlivé hodnoty pak vyjadřují počet značek v daném místě v příslušných značeních M_1 a M_0, a t_1 je vektorem o stejném rozměru, ale s hodnotami vyjadřujícími změnu počtu značek v daném místě po provedení přechodu t_1 (viz vzorec 2-3).

$$M_0 = \begin{pmatrix}1\\0\end{pmatrix}\ M_1 = \begin{pmatrix}0\\1\end{pmatrix}\ t_1 = \begin{pmatrix}-1\\1\end{pmatrix} \qquad \textit{Vzorec 2-3}$$

Incidenční matice P/T Petriho sítě

Nechť $PTN := (P, T, A, AF, M_0)$ je P/T Petriho síť. Incidenční maticí C sítě PTN rozumíme dle Martiník (2015) funkci:

$$\mathrm{C}: (\mathrm{P}\times\mathrm{T}) \longrightarrow Z \qquad \textit{Vzorec 2-4}$$

$$C(p,t) = AF(t,p) - AF(p,t), \text{kde } p \in P, t \in T$$ *Vzorec 2-5*

Příklad incidenční matice *C* P/T Petriho sítě z obrázku 2-5 je uveden ve výpisu 2-3.

Výpis 2-2: Incidenční matice P/T Petriho sítě z obrázku 2-5, zdroj: vlastní

	T1	T2
P1	-3	2
P2	1	-1

2.7.2 P-invarianty P/T Petriho sítě

Vektor *Y* je P-invariantem P/T Petriho sítě PTN, pokud je výsledkem násobení vektoru *Y* a incidenční matice *C* nulový vektor.

$$Y \cdot C = \vec{0}$$ *Vzorec 2-6*

Kardinalita množiny *P* určuje počet dimenzí (složek) vektoru *Y*. Pokud je alespoň jedna složka vektoru *Y* nenulová, jedná se o tzv. **netriviální** P-invariant. Nulový vektor je **triviálním** P-invariantem. Pokud jsou všechny složky vektoru *Y* kladné, jedná se o **pozitivní** P-invariant.

Nechť *PTN* := (*P*, *T*, *A*, *F*, M_0) je P/T Petriho síť, *Y* nechť je p-invariantem sítě PTN. Jestliže v síti PTN existuje sekvence přechodů σ tak, že platí $M_0 [\sigma\rangle M$, pak platí vztah:

$$Y M = Y M_0$$ *Vzorec 2-7*

Pro matematické důkazy viz Martiník (2015).

2.7.3 Vybrané vlastnosti P/T Petriho sítí

V rámci této kapitoly budeme uvažovat P/T Petriho síť *PTN* := (*P*, *T*, *A*, *AF*, M_0) (viz kapitola 2.4.4). P/T Petriho síť PTN je:

Bezpečná

Místo $p \in P$ se označuje za bezpečné, pokud platí:

$$\forall M \in [M_0\rangle: M(p) \leq 1$$ *Vzorec 2-8*

Za předpokladu, že každé místo sítě PTN je bezpečné, je PTN **bezpečná síť**. (Martiník, 2015)

Omezená

Místo $p \in P$ PTN je považováno za k-omezené, pokud platí:

$$\exists k \ \in N_0 \ \forall M \in [M_0\rangle : M(p) \leq k \qquad \textit{Vzorec 2-9}$$

Místo $p \in P$ je omezené, pokud je k-omezené. Pokud je každé místo sítě PTN omezené, je PTN **omezená síť**. (Martiník, 2015)

Živá

PTN je **živá** právě tehdy, když pro každé dosažitelné značení *M* a každý přechod $t \in T$ existuje značení *M'* dosažitelné ze značení *M*, v němž je přechod *t* proveditelný (viz vzorec 2-10)

$$\forall M \in [M_0\rangle \ \ \forall t \in T \ \ \exists M' \ \in \ [M\rangle \ t \ \textbf{\textit{en}} \ M \qquad \textit{Vzorec 2-10}$$

Je-li síť PTN živá, její počáteční značení M_0 nazýváme **živé značení**. (Martiník, 2015)

Slabě živá

Pokud pro každý přechod $t \in T$ platí, že existuje alespoň jedno značení *M*, kde je přechod *t* proveditelný (viz vzorec 2-11), je síť PTN **slabě živá**.

$$\exists M \in [M_0\rangle \ \ \forall t \in T \ \ t \ \textbf{\textit{en}} \ M \qquad \textit{Vzorec 2-11}$$

PTN neobsahuje deadlock

Pokud pro každé dosažitelné značení $M \in [M_0\rangle$ platí, že je proveditelný alespoň jeden přechod $t \in T$, pak síť PTN **neobsahuje deadlock**.

Dobře vytvořená

Za předpokladu, že je síť PTN **omezená** a **živá**, říkáme, že je **dobře vytvořená**.

Reverzibilní

V případě že z každého dosažitelného značení $M \in [M_0\rangle$ sítě PTN může být dosaženo jejího počáteční značení M_0, je síť PTN **reverzibilní**.

Silně propojená

O silně propojené síti PTN hovoříme právě tehdy, když existuje přímá orientovaná cesta mezi každými jejími dvěma místy $p_1 \in P$ a $p_2 \in P$, nebo přechody $t_1 \in T$ a $t_2 \in T$.

2.7.4 Graf dosažitelných značení P/T Petriho sítě

Pomocí grafu dosažitelných značení lze stanovit, jestli je dané značení *M* P/T Petriho sítě PTN dosažitelné. Poté hovoříme o dosažitelném značení *M* (zatím jsme žádné jiné neuvažovali). Grafem dosažitelných značení PTN rozumíme uzlově a hranově ohodnocený orientovaný graf, jehož hrany jsou ohodnoceny přechody dané sítě a uzly ohodnoceny jednotlivými značeními *M* dosažitelnými z jejího počátečního značení M_0. Nevýhodou grafu dosažitelných značení PTN je však obecně příliš vysoká složitost grafu a komplikovanost jeho stanovení. (Reisig, 2013) (Martiník, 2015)

Nechť *PTN* := (*P*, *T*, *A*, *AF*, M_0) je PTN. Jejím grafem dosažitelných značení *G* rozumíme uspořádanou čtveřici $G := ([M_0\rangle,\ H,\ T,\ \varphi)$, kde:

- $(M_1, M_2) \in H \Leftrightarrow \exists t \in T: M_1[t\rangle M_2$, kde $M_1, M_2 \in [M_0\rangle$,
- $\varphi(M_1, M_2) = t \Leftrightarrow\ M_1[t\rangle M_2$, kde $M_1, M_2 \in [M_0\rangle, t \in T$. (Martiník, 2015)

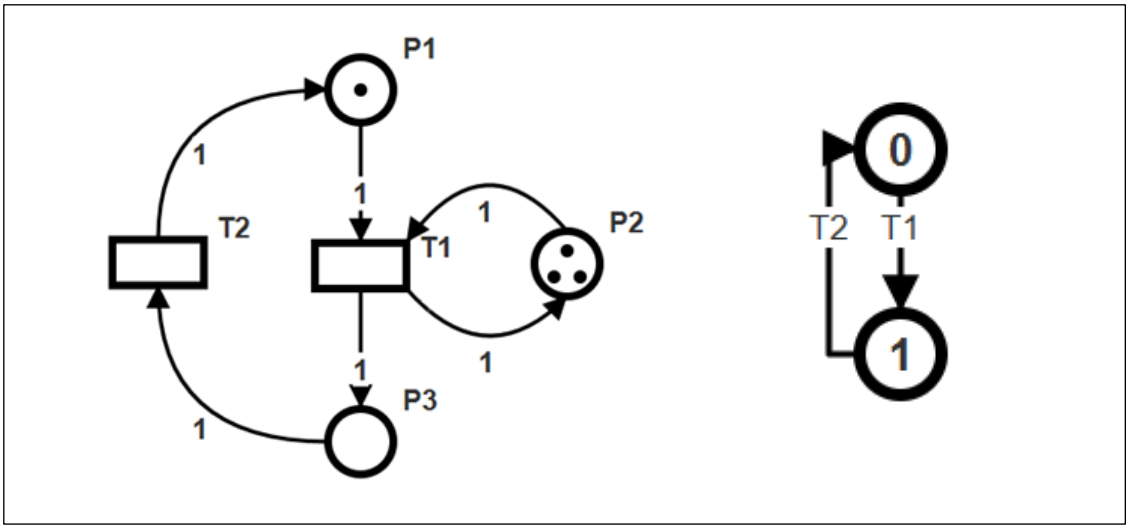
Dle Reisig (2013) lze na základě vlastností grafu dosažitelných značení *G* zjistit, zda má P/T Petriho síť PTN následující vlastnosti (viz kapitola 2.7.6):

- **PTN neobsahuje deadlock** – každý uzel grafu *G* je počátečním uzlem alespoň jedné hrany,
- **PTN je živá** – pro každý přechod $t \in T$ začíná v každém uzlu grafu *G* cesta *C*, pro kterou platí, že je v jejím rámci přechod *t* proveditelný (jedna z hran cesty *C* je ohodnocená přechodem *t*),
- **PTN je slabě živá –** v grafu *G* existuje hrana ohodnocená přechodem $t \in T$,
- **PTN je omezená** – graf *G* je konečný (tj. obsahuje konečný počet uzlů),
- **PTN je reverzibilní** – graf *G* je silně propojeným grafem. (Reisig, 2013)

Fatálním problémem grafu dosažitelných značení (dále jen GDZ) je skutečnost, že počet jeho uzlů je nekonečný, pokud v PTN existuje neohraničené místo. Abychom se vyvarovali nekonečného grafu dosažitelných značení dané P/T Petriho sítě PTN, je nutné konstruovat její tzv. **graf pokrytí**. (Reisig, 2013)

Konstrukce sekvenčního grafu dosažitelných značení

P/T Petriho síť PTN, pro kterou byl v rámci této kapitoly zpracován GDZ, je uvedena i s odpovídajícím GDZ na obrázku 2-6.



Obrázek 2-6: Vzorová PTN a její GDZ, zdroj: vlastní

Pro počáteční značení této P/T Petriho sítě PTN platí, že místu P1 náleží 1 značka a místu P2 náleží 3 značky. Pomocí GDZ lze zjistit, jaká značení jsou pro uvedenou P/T Petriho síť PTN dosažitelná. Dosažitelná značení jsou v GDZ reprezentována pomocí uzlů **1** a **2**, hrany grafu pak reprezentují provedení přechodů *T1* a *T2*.

GDZ lze popsat pomocí následujících matematických vztahů:

$$M_0 = (1,3,0), M_1 = (0,3,1) \qquad \textit{Vzorec 2-12}$$

$$M_1 \xrightarrow{T2} M_0, M_0 \xrightarrow{T1} M_1 \qquad \textit{Vzorec 2-13}$$

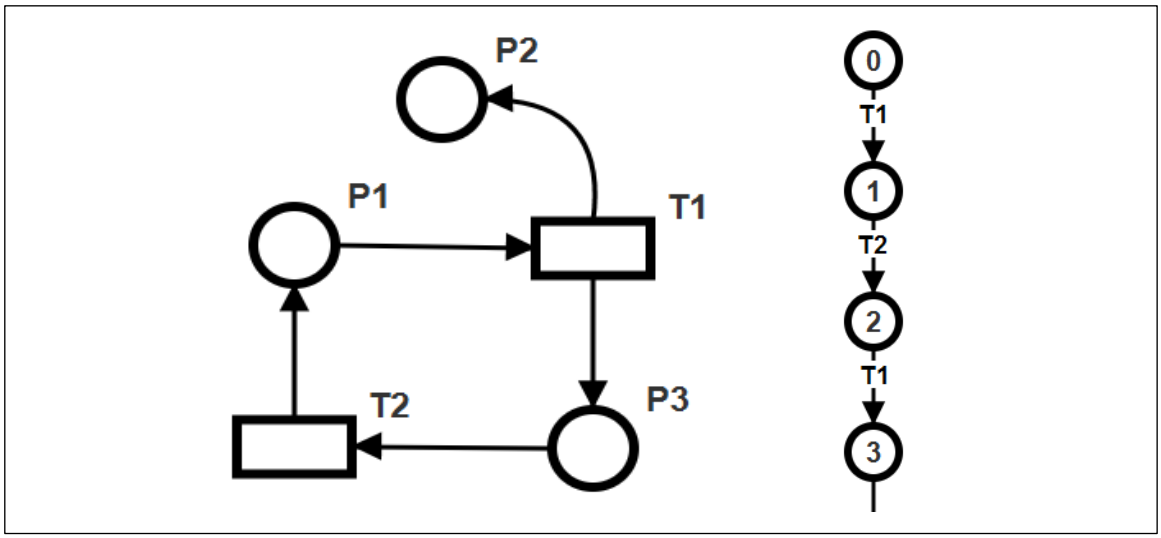
2.7.5 Graf pokrytí

Výsledek procedury tvorby grafů pokrytí z příslušných grafů dosažitelných značení není jednoznačně definován a může vniknout více odlišných grafů pokrytí. Pro účely konstrukce grafu pokrytí P/T Petriho sítě PTN je zavedena speciální hodnota počtu značek náležejících v daném značení *M* místu $p \in P$, a to ω (viz vzorec 2-14) – značení *M* P/T Petriho sítě PTN je zde zobecněno do podoby tzv. ω-značení.

$$M_1 = (1,1,\omega)$$ *Vzorec 2-14*

Konstrukce grafu pokrytí

Pro účely zpracování grafu pokrytí byla vybrána P/T Petriho síť PTN uvedená na obrázku 2-7.



Obrázek 2-7: Vzorová PTN a její GDZ, zdroj: vlastní

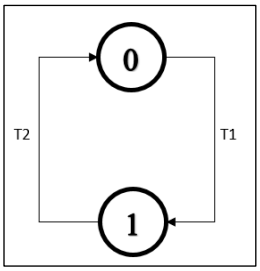
GDZ (viz obr. 2-8) této sítě PTN je nekonečný, pokud v jejím počátečním značení M_0 náleží alespoň 1 značka místu P1 nebo P3. Za předpokladu, že se v počátečním značení M_0 nachází právě 1 značka v místě P1, GDZ sítě má podobu nekonečné sekvence provádění přechodů T1 a T2. Počty značek ve značeních dosažitelných z počátečního značení M_0 nabývají hodnot uvedených ve vzorci 2-15.

$$\mathrm{M}_0 = (1,0,0), \mathrm{M}_1 = (0,1,1), \mathrm{M}_2 = (1,1,0), \mathrm{M}_3 = (0,2,1)$$ *Vzorec 2-15*

Na základě opakujícího se vzoru (1, x, 0) nebo (0, x, 1) v jednotlivých značeních sítě lze definovat dvě ω-značení (viz vzorec 2-16).

$$\mathrm{M}_0 = (1,\omega,0), \mathrm{M}_1 = (0,\omega,1),$$ *Vzorec 2-16*

Graf pokrytí (viz obr. 2-8) lze tedy konstruovat na základě těchto dvou ω-značení, od počtu značek v místě P2 bylo abstrahováno. Ze značení M_0 lze přejít do značení M_1 provedením přechodu T1 a zpět do značení M_0 provedením přechodu T2.



Obrázek 2-8: Graf pokrytí vzorové PTN z obr. 2-7, zdroj: vlastní

2.8 Procesní P/T Petriho sítě

Procesní P/T Petriho sítí, dále jen PPTN, rozumíme uspořádanou 10-tici $PPTN := (P, T, A, AF, TP, RP, IP, OP, RM_s, M_0)$, kde:

- P je konečná neprázdná množina **míst,**
- T je konečná neprázdná množina **přechodů,** $P \cap T = \emptyset$,
- $A \subseteq (P \times T) \cup (T \times P)$ je konečná množina **hran**,
- AF: $(P \times T) \cup (T \times P) \rightarrow \boldsymbol{N_0}$ je **hranová funkce**,
- $AF(x, y) > 0 \Leftrightarrow (x, y) \in A$, $AF(x, y) = 0 \Leftrightarrow (x, y) \notin A$, kde $x, y \in P \cup T$,
- TP je funkce priorit přechodů, TP: $T \rightarrow N$,
- RP je konečná množina **míst zdrojů**, $RP \subset P$,
- IP je **vstupní místo**, $IP \in (P \setminus RP)$, tj. jediné místo, pro které platí: $\bullet IP = \emptyset$,
- OP je **výstupní místo**, $OP \in (P \setminus RP)$, tj. jediné místo, pro které platí: $OP\bullet = \emptyset$,
- RM_s je množina **povolených statických značení sítě**, $RM_s \subseteq \mathbf{M_s}$, kde $\mathbf{M_s}$ je množina všech statických značení M_s sítě $PPTN$,
- M_0: $P \rightarrow \boldsymbol{N_0}$ je **počáteční značení**,
- (P, T, A) je souvislá síť. *(Martiník, 2015)*

PPTN jsou podtřídou P/T Petriho sítí se zavedením priorit přechodů (viz kapitola 2.8.1). Pomocí vstupního a výstupního místa (viz kapitola 2.8.2) je dle Huang (2012) definováno rozhraní procesní PTN sítě.

2.8.1 Funkce priorit přechodů

V P/T Petriho síti je výběr jednoho z proveditelných přechodů k realizaci v rámci jejího sekvenčního běhu čistě náhodný (popřípadě jej zvolí uživatel, provádějící simulaci). Tento problém je řešen v případě PPTN zavedením tzv. funkce priorit přechodů. Každému přechodu $t \in T$ je přiřazeno přirozené číslo vyjadřující jeho prioritu – pokud je v daném značení M PPTN proveditelných více přechodů, je proveden přechod s nejvyšší prioritou přechodu.

2.8.2 Typy míst PPTN

Místa PPTN dělíme následovně:

- **obyčejné místo**,
- **místo zdrojů** – jediná místa, která mohou obsahovat značky v tzv. statickém značení M_S PPTN, ve schématu sítě je zakreslujeme formou kružnice se zdvojeným okrajem,
- **vstupní místo (IP) –** každá PPTN obsahuje právě jedno vstupní místo s více nebo žádnou značkou ve vstupním a aktuálním značení a žádnou značkou ve statickém a výstupním značení,
- **výstupní místo (OP) –** každá PPTN obsahuje právě jedno výstupní místo se značkami ve výstupním nebo aktuálním značení a žádnou značkou ve statickém nebo vstupním značení.

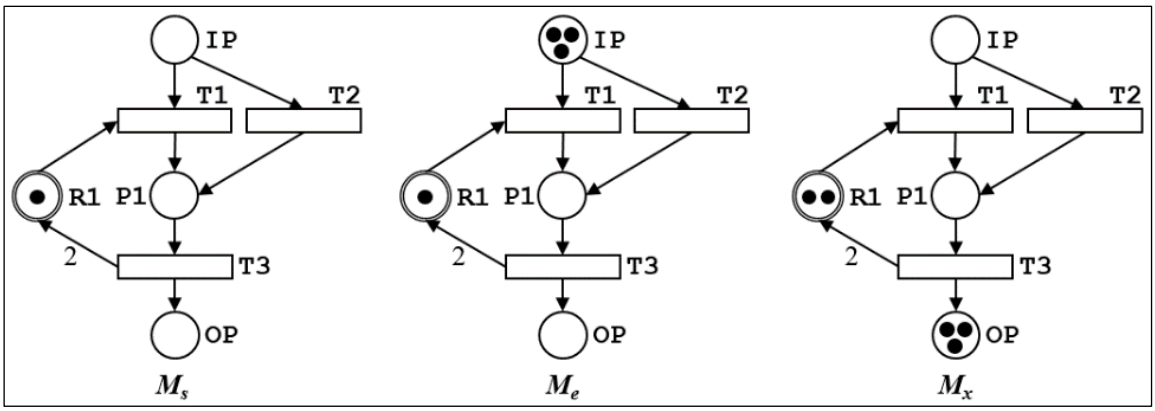
2.8.3 Značení v procesních PTN

V rámci P/T Petriho sítí byly definovány dva typy značení – počáteční značení M_0 a aktuální značení M (viz kapitola 2.5.4). Pro účely PPTN dále zavádíme tzv. statické značení M_S, v němž se jednotlivé značky smí nacházet pouze v některém z míst zdrojů dané PPTN a dále nesmí být v tomto statickém značení M_S proveditelný žádný z přechodů $t \in T$ dané PPTN.

V závislosti na stavu dané PPTN a jejím odpovídajícím značení se mohou v místech sítě nacházet značky dle tabulky 2-1 a obrázku 2-9.

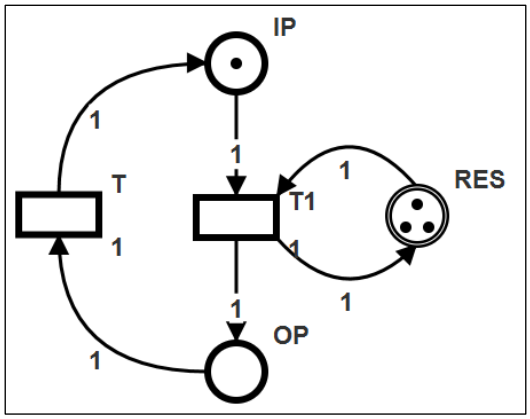
Tabulka 2-1: Počet značek v místech, dle typu míst v závislosti na značení, zdroj: vlastní

Značení	Obyčejná místa	IP	Místa zdrojů	OP
Vstupní M_e	0	>0	≥0	0
Statické M_s	0	0	≥0	0
Aktuální M	≥0	≥0	≥0	≥0
Výstupní M_x	0	0	≥0	≥0



Obrázek 2-9: Příklady PPTN v rozdílných značeních, zdroj: Martiník (2016)

Na obrázku 2-10 je zobrazena PPTN s grafickým znázorněním priorit přechodů a hodnot hranových funkcí.



Obrázek 2-10: Vzorová PPTN s prioritami přechodů, zdroj: vlastní

2.9 Property-Preserving Petri Net Process Algebra

Procesu zachování vlastnosti je věnována vědní disciplína zvaná *Property-Preserving Petri Net Process Algebra* (PPPA) dále jen procesní algebra.

V rámci procesní algebry je dle Huang (2012) definováno pět tříd tzv. operátorů, jejichž pomocí lze pracovat s komponentami (viz kapitola 2.9.1) nebo danou PPTN modifikovat. Třídy operátorů jsou následující:

- operátory rozšíření,
- operátory kompozice,
- vylepšení zachovávající vlastnosti,
- redukce zachovávající vlastnosti,
- operátory sloučení míst pro sdílení zdrojů.

Pro více informací viz Huang (2012).

2.9.1 Architektura založená na komponentách

PPTN se v praxi skládají ze znovupoužitelných komponent. Vlastnosti PPTN však po implementaci těchto komponent nemusí být zachovány – žádané vlastnosti můžou být ztraceny, nebo naopak se mohou objevit nové, nežádoucí.

2.10 Procesy

Proces je posloupnost činností, která se v zásadě pravidelně opakuje, resp. není určena k jednorázovému využití. Složkami procesu jsou jak zmíněné činnosti, tak jeho vstupy, výstupy, rozhraní, vlastník, zainteresované osoby, zákazníci a další.

Modelování procesu

Proces je možné definovat a podrobně modelovat vyjma Petriho sítí například následujícími způsoby:

- obyčejným diagramem toků (Vlček a Kalužová, 2012),
- metodikou **BPM** (Business Process Management) a jeho grafické notace BPMN (Salatino a Aliverti, 2008),
- metodikou **DEMO** (Design and Engineering Methodology for Organizations) (Dietz, c2006),
- metodikou **REA** (Resource Event Agent) (Hrubý, 2010).

Simulace procesu

Dle Fiala a Ministr (2013, s. 19) „*Simulace procesů představuje analýzu, která sleduje změny procesu v čase. Je zaměřena na dynamické parametry procesů, které jsou nejčastějším předmětem zájmu při zdokonalování procesů.*“ Simulaci lze chápat i jako levný způsob, jak vyzkoušet nově definovaný proces, ještě před jeho ostrým provozem.

Modelování procesů pomocí Petriho sítí

Modelování procesů pomocí P/T Petriho sítí je vysoce náročné. Obyčejné P/T Petriho sítě nemají totiž zavedeny explicitní logické operátory jako *and*, *or*, popřípadě *xor*. Dále není uvažován čas a identita značky – značky jsou anonymní.

Předmětem této práce jsou však zejména PPTN, které lze považovat za minimální nástroj pro modelování procesů. Jak bylo uvedeno v kapitole 2.8, zahrnují totiž speciální místa a také prioritní funkci přechodů.

Modelovat procesy lze dle Van Hee et al. (2013) také například prostřednictvím třídy Business Process Petri Nets. Obdobou prioritní funkce přechodů jsou zde inhibiční a reset hrany.

3 Analýza dostupných aplikací pro simulaci Petriho sítí a využité technologie

Nedílnou součástí vývoje softwaru je i analýza dostupných produktů podobného charakteru. I když je vyvíjená aplikace unikátní svou portabilitou, dostupností i bez internetového připojení a specifickým zaměřením na třídu PPTN, je nutné kriticky zhodnotit alespoň několik vybraných aplikací v této oblasti. Analýze dostupných aplikací je tedy věnována první polovina této kapitoly.

Technologie, postupy a základní poznatky nutné pro vývoj nové multiplatformní aplikace jsou blíže definovány v druhé části této kapitoly.

3.1 Dostupné aplikace

V této kapitole jsou jednotlivé aplikace blíže analyzovány z hlediska funkcionality (v rámci základních Petriho sítí), technologií, nad kterými jsou implementovány, zaměření na danou podtřídu Petriho sítí a jejich licencování.

Co se týče metodiky výběru je pak většina vybraných aplikací uvedena v seznamu nástrojů pro modelování Petriho sítí na webových stránkách *www.informatik.uni-hamburg.de*. Ve všech případech jde o plně freewarové produkty nebo volně využitelné alespoň pro nekomerční a akademické účely. Aplikace se liší množstvím nabízených funkcionalit i technologiemi. Důležitým kritériem při výběru bylo i to, že je aplikace dostupná v anglickém jazyce a neobsahuje reklamy.

Předmětem analýzy není hodnocení ani porovnání, jde jen o přehled v současné době využitelných aplikací s možností modelování základních, popřípadě modifikovaných P/T Petriho sítí, což je rovněž cílem vyvíjené aplikace.

V této oblasti je dostupné také množství vysoce specializovaných nástrojů jako například:

- **ORIS Tool** – moderní a udržovaný produkt určený pro časované a stochastické Petriho sítě,
- **PetriSim 5** – nástroj s bohatými možnostmi konfigurace všech komponent Petriho sítí, založený na časovaných Petriho sítích, z pohledu současných grafických uživatelských rozhraní však extrémně zastaralý.

3.1.1 CPN Tools

CPN Tools je dle webových stránek (http://cpntools.org/) nástroj pro editování, simulaci a analýzu třídy barevných Petriho sítí. Aplikace dále umožňuje inkrementální kontrolu syntaxe a generování zdrojového kódu již při návrhu sítě. Pomocí této aplikace lze efektivně simulovat jak časované, tak nečasované sítě. Z hlediska vlastností lze zkoumat například ohraničenost nebo živost sítě. (CPN Tools, c2017)

Nástroj byl původně vyvíjen CPN Group na Aarhuské univerzitě v letech 2000 až 2010. Od roku 2010 byl vývoj přesunut na Eindhovenské Universitě Technologií v Nizozemí. (CPN Tools, c2017)

Architektura a licencování

CPN Tools je licencován na základě jeho komponent:

- **grafické uživatelské rozhraní** je licencováno dle GNU General Public License (GPL) verze 2,
- **simulátor** je licencován dle GNU General Public Licence (GPL) a 4 bodové BSD (Berkley Software Distribution),
- **modul Access/CPN** je licencován dle GNU Lesser General Public Licence (LGPL) verze 2.1. (CPN Tools, c2017)

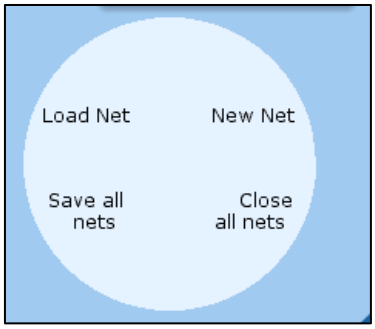
Dále dle stránek CPN Tools již autoři postupně vydávají zdrojový kód, s tím, že cílem je zřejmě učinit CPN Tools open-source produktem.

Uživatelské grafické rozhraní programu

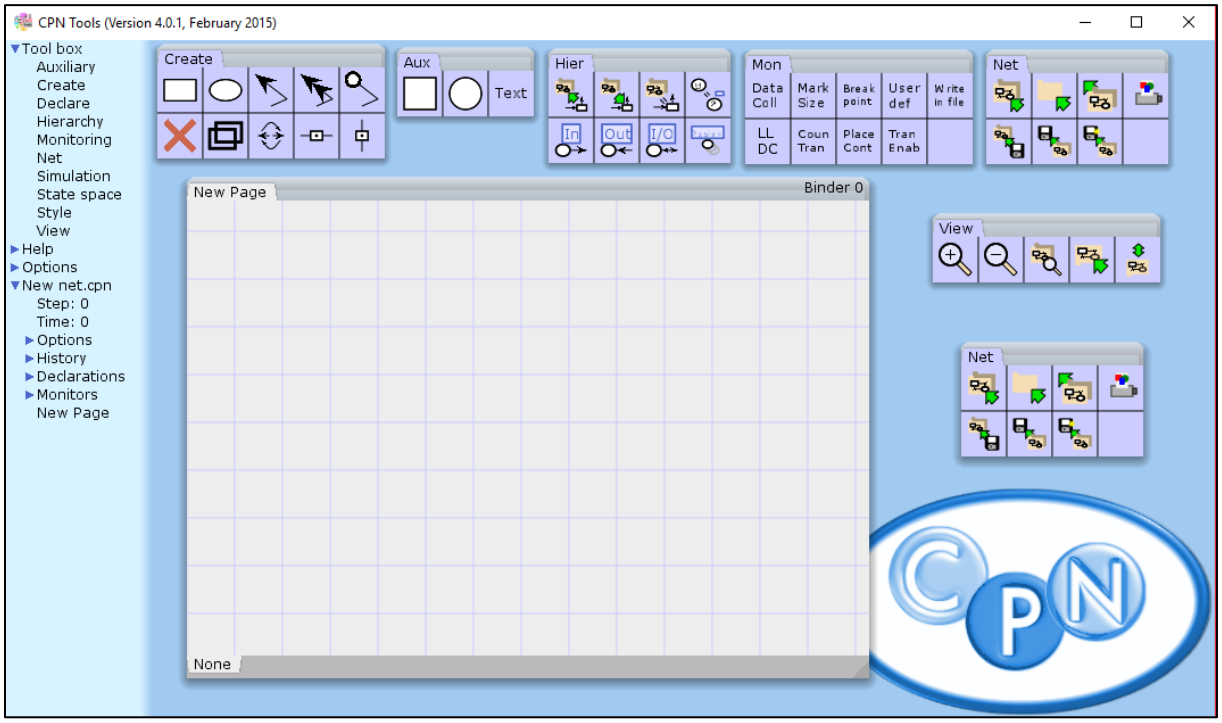
Uživatelské grafické rozhraní (GUI) je rozvrženo dle současných trendů nevšedně, a to na levý panel s drobným textem bez jakéhokoliv náznaku ikon, přičemž panel připomíná víceúrovňový neuspořádaný seznam. Po kliknutí a přetažení textu doleva do druhé části GUI lze vytvořit a zároveň umístit odpovídající ovládací panel, přičemž tento postup lze opakovat a tímto způsobem umístit více stejných ovládacích panelů, které jsou nazvány „Binder“. Rozbalením úrovně seznamu se celý panel přizpůsobí velikosti nejširšího popisku a v případě, že se blízko panelu nachází ovládací panely, popisek nelze přečíst.

Umístění panelu pro kreslení samotné sítě je řešeno podobným způsobem, až na to, že je lze vytvořit a umístit po kliknutí pravým tlačítkem myši a vybráním odpovídající akce z „kruhové nabídky“ (viz obr. 3-1).

Kreslení sítě probíhá tak, že uživatel vybírá prvky z volně umístěných panelů a poté kliká na kreslící plochu, popřípadě může prvky přesunovat nebo propojovat zcela intuitivním způsobem (viz obr. 3-2).



Obrázek 3-1: Kruhová nabídka aplikace CPN Tools, zdroj: vlastní

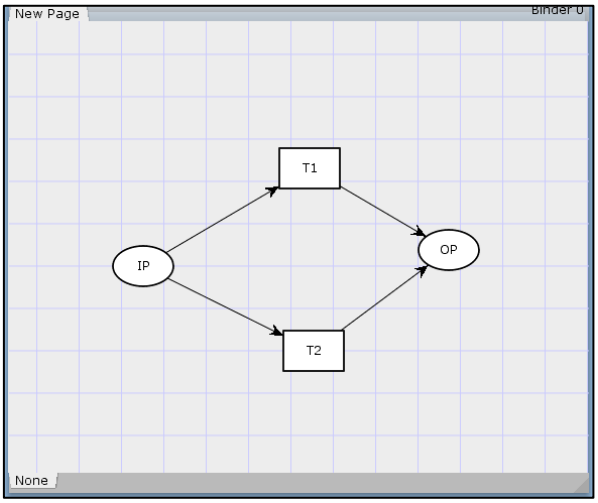


Obrázek 3-2: Hlavní GUI aplikace CPN Tools, zdroj: vlastní

Funkcionalita

V rámci CPN Tools lze definovat nepřeberné množství dodatečných vlastností sítí, a tím modelovat velké množství přesně specifikovaných problémů, které lze řešit

pomocí barevných Petriho sítí, tj. vysokoúrovňových Petriho sítí, v nichž lze definovat více typů rozlišitelných značek prostřednictvím jejich tzv. obarvení.



Obrázek 3-3: Petriho síť, nakreslená v aplikaci CPN Tools, zdroj: vlastní

Technologie

CPN Tools je dle analýzy souborů ve složce jeho instalace postaven na několika technologiích:

- programovacím jazyce **Java,**
- programovacím jazyce **C/C++,**
- popřípadě ostatních**.**

Programové moduly jsou na sobě nezávislé a mají podobu spustitelných souborů s příponou jar nebo exe.

Jako problém autor této práce považuje komunikaci programu na pozadí, na kterou není uživatel aplikace vůbec upozorněn. Jde o spouštění softwarových serverových komponent programu, ke kterému nedochází okamžitě po spuštění, ale v průběhu práce. Toto chování bylo vyhodnoceno antivirovým programem za nebezpečné (zcela regulérně) a CPN Tools byl následně přesunut do virového trezoru.

Portabilita

Současná verze aplikace 4.0.1 je bohužel zcela neportabilní a je určena pouze pro operační systém Microsoft Windows. Uživatelé operačních systémů Linux nebo

Mac aplikaci musí spouštět v rámci virtuálního stroje nebo stáhnout již neudržovanou verzi.

Vývoj verze pro mobilní zařízení dle autora této práce nepřipadá v úvahu z hlediska grafického uživatelského rozhraní ani principu modularity, což jsou zcela oddělené komponenty, které komunikují na pozadí.

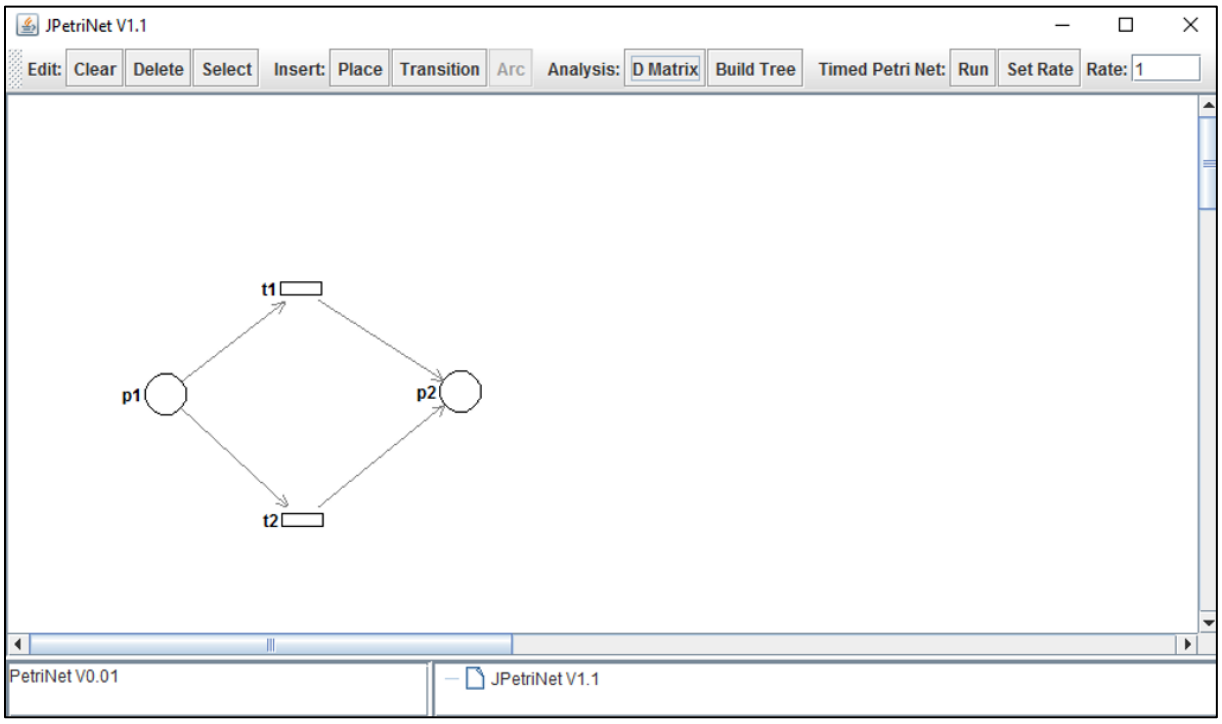
3.1.2 JPetriNet

JPetriNet je představitelem open-source portabilních aplikací napsaných v programovacím jazyce Java. Jde o aplikaci se základní funkcionalitou a minimalistickým uživatelským prostředím.

Uživatelské grafické rozhraní

Rozhraní (viz obr. 3-4) je složeno z následujících částí:

- **lišta s ovládacími prvky** – nachází se zde tlačítka týkající se editace, tvorby, analýzy i simulace Petriho sítě zároveň, jde o veškerou funkcionalitu programu,
- panel pro **kreslení sítí**,
- panel s **verzí programu**.



Obrázek 3-4: Hlavní GUI aplikace JPetriNet, zdroj: vlastní

Funkcionalita

Prostřednictvím JPetriNet lze modelovat a simulovat základní P/T Petriho sítě. Místa mohou obsahovat více značek jednoho typu. Z vlastností Petriho sítí lze analyzovat incidenční matici a kauzální síť. Aplikaci lze stáhnout z webových stránkách projektu http://jpetrinet.sourceforge.net.

Technologie

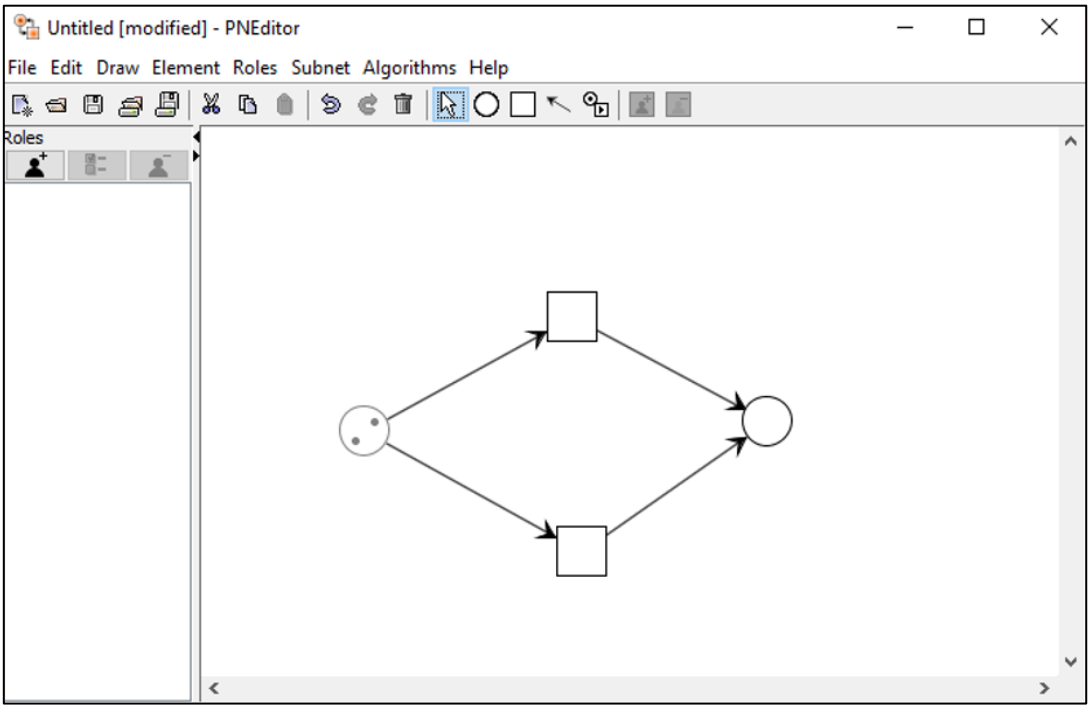
Aplikace je napsána výhradně v programovacím jazyku Java. Instalace není vyžadována, jediným souborem je soubor typu jar, pomocí něhož se aplikace také spouští.

Portabilita

Vzhledem k využitým technologiím je JPetriNet portabilní mezi desktopovými platformami. O verzi pro mobilní zařízení by bylo možné uvažovat, kdyby bylo GUI programu přepsáno do technologie JavaFX.

3.1.3 PNEditor

Další z řady programů umožňujících modelování a simulaci základních P/T Petriho sítí. Jako v předchozím případě se jedná o software napsaný v programovacím jazyce Java s licencí GNU GPL verze 3.



Obrázek 3-5: Hlavní GUI aplikace PNEditor, zdroj: vlastní

Grafické uživatelské rozhraní

V případě programu PNEditor lze hovořit o tradičním rozvržení GUI (viz obr. 3-5), které se skládá z následujících částí:

- lišta s nabídkou,
- lišta rychlého přístupu k základním funkcím,
- po levé straně seznam rolí,
- po pravé straně prostor pro kreslení sítě.

Funkcionalita

V rámci aplikace lze modelovat a simulovat P/T Petriho sítě o jednom typu značky. Místa mohou být statická, v terminologii diplomové práce je lze chápat jako místa zdrojů. Dále jsou podporovány podsítě a definice rolí. Petriho síť je možno ve výsledku exportovat například jako obrázek formátu PNG.

Z hlediska analýzy však lze zkoumat pouze omezenost sítě.

Aplikaci lze stáhnout z webových stránek http://www.pneditor.org.

Technologie

Aplikace je stejně jako v předchozím případě založena na technologii Java 6 a spouští se pomocí jediného souboru typu jar.

Portabilita

Vzhledem k využitým technologiím je aplikace portabilní mezi desktopovými platformami. Vývoj verze pro mobilní zařízení by vyžadoval přepracování lišty s nabídkou a využití JavaFX.

3.1.4 TAPAAL

Programový systém TAPAAL: Tool for Verification of Timed-Arc Petri Nets (Nástroj pro verifikaci časovaných Petriho sítí) je produktem Aalborgské Univerzity. Poslední verze 3.2.1 byla vydána 19.9.2016, software je tedy aktivně podporován. (TAPAAL, c2017)

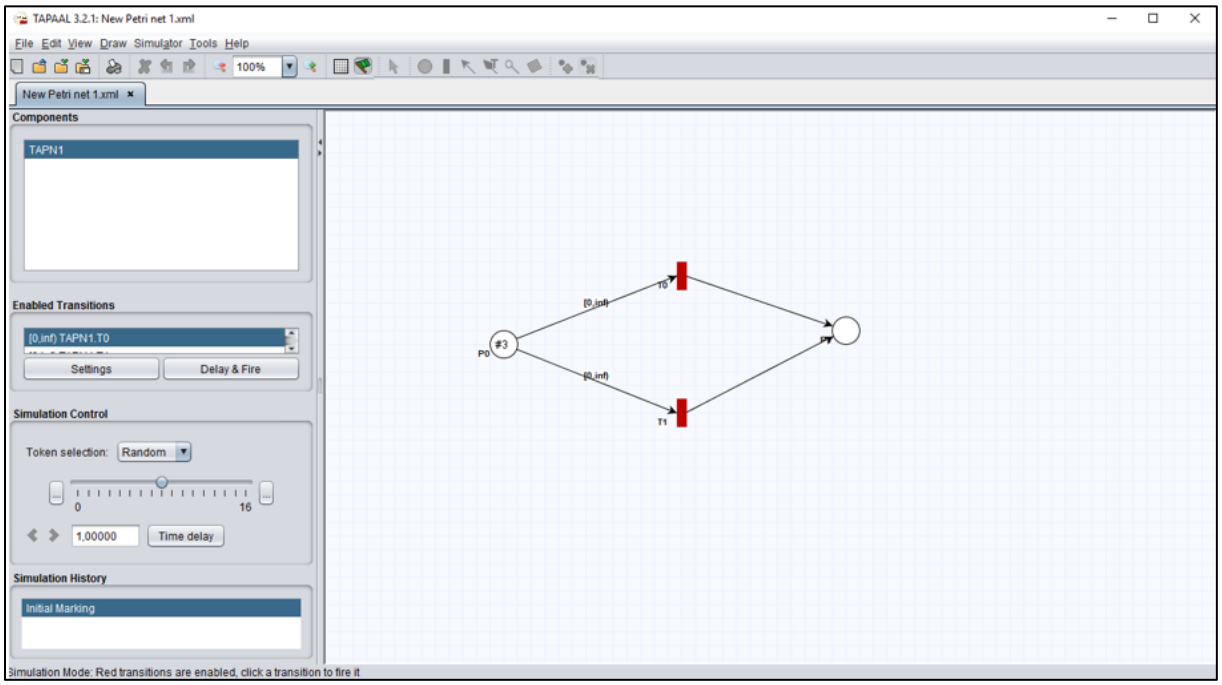
Architektura a licencování

Programový systém TAPAAL se skládá z několika oddělených programových modulů, způsob licencování je následující:

- **GUI** – Open Source Licence 3.0,
- **moduly aplikační logiky a verifikace** – BSD licence,
- **engine kontinuální validace** – GPL licence verze 2.

Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (viz obr. 3-6) je rozvrženo stejně jako v případě aplikace PNEditor, s tím rozdílem, že se po levé straně nachází ovládací panel vybraného režimu, tj. základní režim a režim simulace.



Obrázek 3-6: Hlavní GUI apliace TAPAAL, zdroj: vlastní

Funkcionalita

Jak bylo uvedeno v úvodu kapitoly tohoto nástroje, jeho hlavním účelem je modelování a simulace časovaných Petriho sítí, využití pro základní sítě však není vyloučeno. Místa se dělí na obyčejná a sdílená. Prostřednictvím časovaných hran lze přesouvat zvolený počet značek a nastavovat její časový interval.

Režim simulace je interaktivní, uživatel má možnost v rámci simulace rozhodovat o tom, jaký přechod se provede v případě kolize. Dále je k dispozici konfigurace a historie.

V rámci analýzy vlastností sítě je možné zjišťovat například ohraničenost sítě, živost, příp. vygenerovat statistiky sítě (počet míst, přechodů, atd.).

Aplikaci lze stáhnout z webových stránek http://www.tapaal.net/.

Technologie

TAPAAL funguje na podobném principu jako CPN Tools. GUI je vyvíjeno v programovacím jazyku Java 7, zatímco moduly aplikační logiky mají podobu nezávislých spustitelných programů (typu exe).

Portabilita

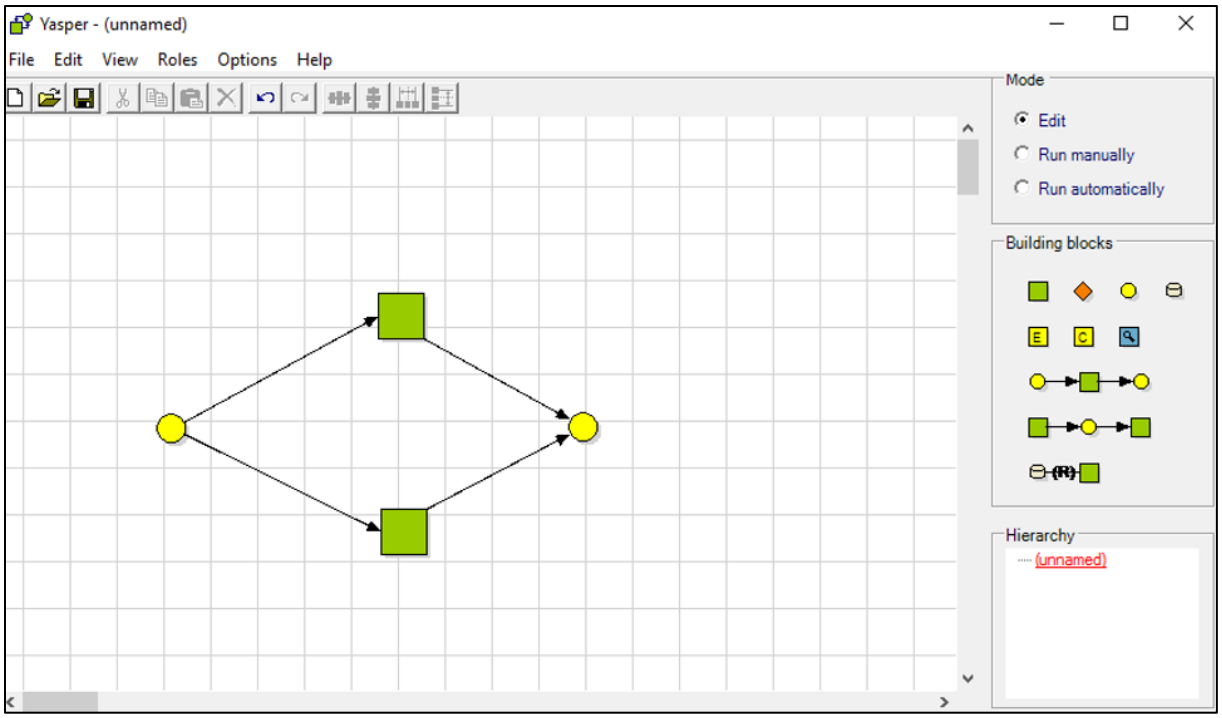
Nástroj TAPAAL je možné spouštět na všech desktopových platformách s tím, že na platformě Linux a Mac je k dispozici ve 32 i 64-bitové verzi. Tvorba mobilní verze tohoto programu by byla stejně jako v případě CPN Tools náročná kvůli architektuře programu.

3.1.5 Yasper

Yet Another Smart Proces Editor – Yasper je nástrojem pro simulaci workflow procesů pomocí rozšířených Petriho sítí, vyvinula jej technická univerzita Eindhoven ve spolupráci s Deloitte v roce 2005. Licence není na webových stránkách projektu http://www.yasper.org uvedena.

Grafické uživatelské rozhraní

Minimalistické grafické rozhraní (viz obr. 3-7) umožňuje mimo jiné výběr režimu (editace, ruční simulace, automatická simulace) a přehledný panel se stavebními bloky Petriho sítě.



Obrázek 3-7: Hlavní GUI aplikace Yasper, zdroj: vlastní

Funkcionalita

Dle jeho autorů je nástroj určen k modelování a simulaci workflow procesů tak, že je přechod brán jako krok v procesu a místo jako podmínka uskutečnění kroku. Jde o jednoduchý a intuitivní nástroj, v jehož rámci však není nabízena vůbec žádná funkcionalita pro analýzu vlastností Petriho sítě. Sítě lze exportovat jako obrázek nebo ve formátu diagramu pro MS Visio.

Architektura a technologie

Aplikace je vyvinuta v prostředí .NET 2.0 a toto prostředí je nutno nainstalovat. Aplikace nemá modulární charakter.

Portabilita

Software vyvinutý pro virtuální stroj .NET je možno používat výhradně na zařízení s operačním systémem MS Windows. Na ostatních desktopových platformách je nutné jej spouštět pomocí virtualizace OS.

3.2 Technologie a nástroje využité ve vývoji aplikace

V této kapitole jsou definovány technologie implementace vyvíjené aplikace a cílové platformy.

3.2.1 Java

Java je objektově orientovaný programovací jazyk s funkcionálními prvky v současné době ve verzi 8, nicméně je již plánovaná verze 9. První verze jazyka a knihoven tříd pochází z roku 1995. Programy napsané v jazyce Java nejsou kompilovány přímo do strojového kódu, nýbrž do tzv. bytekódu, který je následně kompilován Just-In-Time (JIT) kompilátorem. Výhodou této technologie je možnost dynamické optimalizace kódu a menší velikost programů, její nevýhodou je čas potřebný k dynamické kompilaci programových jednotek. Ke správě operační paměti je dedikován Garbage Collector. Běh samotného programu je zajišťován pomocí virtuálního stroje Java Virtual Machine (JVM), což zajišťuje úplnou nebo alespoň částečnou portabilitu napsaných programů mezi platformami.

S příchodem verze 9 je plánováno využití Ahead-Of-Time (AOT) kompilátoru, který má dle Beckwith (2016) zkrátit hlavně čas spouštění programů díky předkompilovaným programovým jednotkám.

Programovací jazyk Java byl vybrán také z důvodu jeho rozšířenosti a zároveň by neměl být problém sdílet a vyvíjet například modul simulace Petriho sítí (viz kapitola 4.2.1) pro více aplikací založených na stejných technologiích.

3.2.2 Framework JavaFX

Programovací jazyk Java byl přizpůsoben již od svých počátků, pokud pomineme Projekt Star7, hlavně pro vývoj webových aplikací, a to jak pro účely zabezpečení back-endu (kódu běžícího na aplikačních serverech), tak multiplatformních appletů běžících rovnou v prohlížeči. Desktopové aplikace byly vyvíjeny pomocí knihovny Java Swing.

V tomto jazyku lze vyvíjet i pro mobilní zařízení. Pro tyto účely byla původně využívána knihovna Java ME (Java Micro Edition). V letech 2007 až 2009 pak byla Java masově rozšířena jako hlavní jazyk pro vývoj aplikací pro chytrá mobilní zařízení s operačním systémem Google Android.

Od vývoje appletů bylo v minulých letech upuštěno (přestaly být podporovány moderními prohlížeči), protože byly předmětem neustálého vyhledávání a zneužívání bezpečnostních mezer, což není nic neobvyklého u takto vysoce používaných technologií. Možnosti tvorby appletů pomocí JavaFX budou v rámci kapitoly tedy zcela opomenuty.

Knihovna Java Swing na druhou stranu byla po dlouhou dobu jednou z hlavních možností vývoje desktopových aplikací v rámci technologie programovacího jazyka Java. Výkon a možnosti knihovny (hlavně pokud šlo o vzhled prvků GUI) však postupem času přestal dostačovat a v roce 2007 byl představen framework JavaFX, který byl oficiálně vydán ve verzi 1.0 v roce 2008. (Sharan, 2015)

Framework JavaFX se postupně rozvíjel a nabízel novou funkcionalitu, nejnovější verze 8 pak byla vydána v roce 2014, kdy také oficiálně nahradila knihovnu Java Swing. Tato verze je využita pro vývoj cílové aplikace a nabízí dle autora diplomové práce nejlepší možnosti zpracování uživatelského rozhraní v Javě v rámci celé její historie.

Základní vlastnosti frameworku JavaFX

Největší výhodou frameworku JavaFX je to, že je v rámci vývoje možné využívat všechny dostupné knihovny napsané v Javě (řádově milióny deklarovaných tříd). Dále není nutné vyvíjet přímo v programovacím jazyce Java, neboť jsou podporovány všechny programovací jazyky z ekosystému JVM, například Groovy nebo Scala. V případě programovacího jazyku Scala lze použít i přímo určený framework ScalaFX. (Sharan, 2015)

Nejdůležitější jsou samotné funkcionality frameworku:

- tvorba uživatelského rozhraní ve formátu XML (modifikace FXML) a jeho stylování pomocí kaskádových stylů,
- bohaté možnosti vázání dat,
- podpora multimédií,
- podpora webového prohlížeče a webového engine,
- výběr z mnoha vestavěných a předvolených animací.

Vybrané funkcionality jsou blíže popsány v následujících odstavcích. (Sharan, 2015)

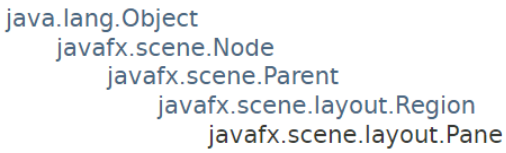
Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je definováno v rámci frameworku JavaFX, na rozdíl od jeho předchůdce Java Swing a mnoha ostatních starších frameworků (.NET Forms, Delphi atd.), pomocí XML, resp. rozšíření JavaFX XML (FXML). Proces

tvorby aplikací probíhá tedy podobně jako v případě platformy Google Android. Na první pohled by se mohlo zdát, že jde o princip tvorby webových stránek, ale ve skutečnosti se přístupy liší v oddělení prezentační a obsahové vrstvy. (Sharan, 2015)

Pro lepší pochopení GUI je nutno zavést následující pojmy:

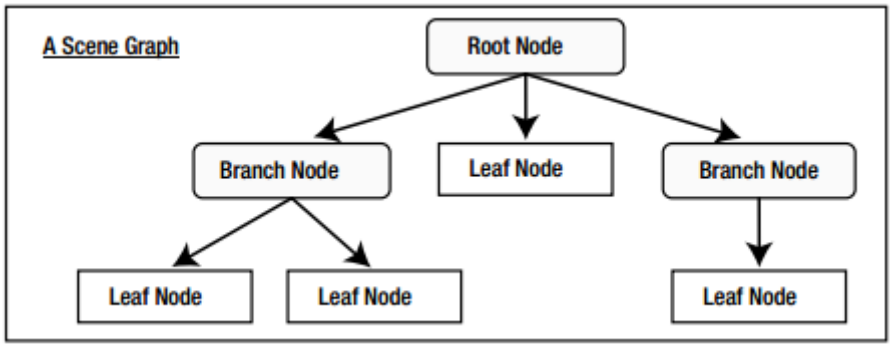
- **stage** (pódium) – fyzické okno aplikace,
- **scene** (scéna) – zobrazovací plocha v rámci pódia,
- **node** (uzel, prvek) – koncový prvek uživatelského rozhraní (popisek, tlačítko, textové pole),
- **pane** (panel, sekce, prostor) – panel, jehož účelem je umožnit vývojáři definovat rozvržení uživatelského rozhraní, může obsahovat další panely nebo listové prvky. Pro příklad bude uveden prostor umožňující ukotvení prvků ke svým hranám (AnchorPane) nebo prostor zobrazující prvky po sebou (VBox). Právě v tomto se návrh GUI liší od návrhu webových stránek,
- **scene graph** (graf scény) – pomocí grafu scény je definována struktura GUI formou stromového grafu viz následující odstavec.



Obrázek 3-8: Hierarchie tříd javafx.scene.Node, zdroj: Dokumentace ORACLE (©2008, 2015)

Uzly stromového grafu scény

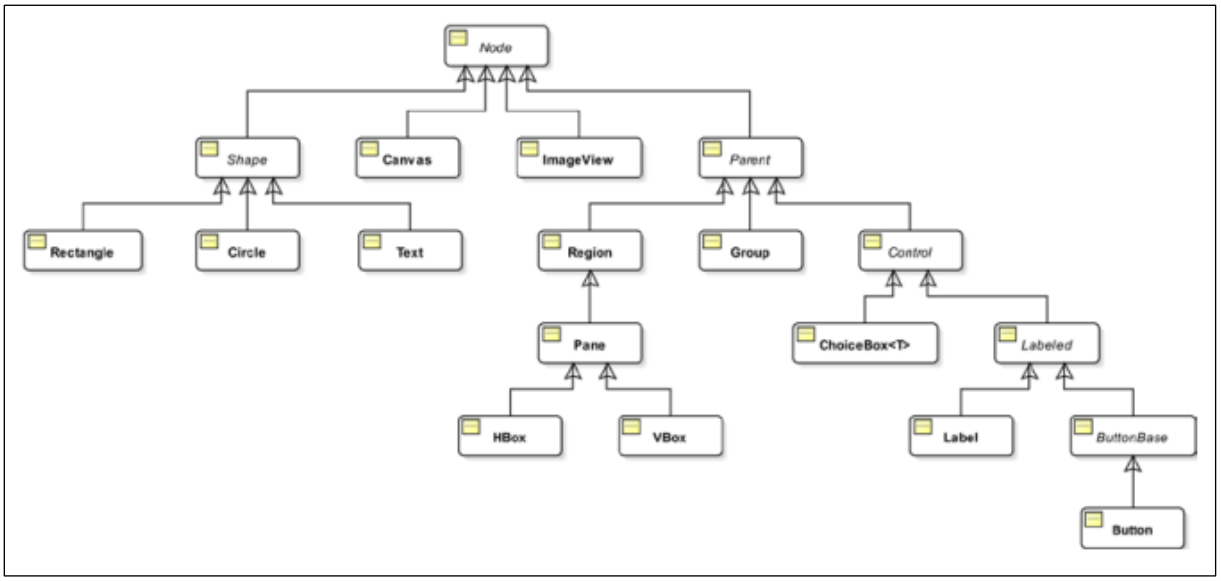
Kořen stromového grafu je klasicky označován jako „root node“ (kořenový uzel, prvek). V rámci definice větví a listů grafu scény je příhodné uvést třídu *javafx.scene.Parent* – dále rodič. Právě podtřídy (například *javafx.scene.layout.Pane*, viz obr. 3-9) mohou tvořit větve (Branch Node) a obsahovat potomky (children), třídy *javafx.scene.Node* a libovolné její podtřídy. Listové uzly jsou reprezentovány třídou *javafx.scene.Node* a libovolné její podtřídy. (Sharan, 2015)



Obrázek 3-9: Graf scény JavaFX, zdroj: Sharan (2015, s. 149)

Hierarchie tříd grafického uživatelského rozhraní

Při pohledu na obrázek 3-10 (diagram vybraných tříd v rámci celkové hierarchie) je jasné, že všechny třídy prvků GUI jsou podtřídami třídy *javafx.scene.Node*. Dvěma hlavními podtřídami jsou *javafx.scene.Shape* a *javafx.scene.Parent*. Dále je zjevná pozice *javafx.scene.layout.Pane* a *javafx.scene.Control*. (Sharan, 2015)



Obrázek 3-10: Hierarchie tříd javafx.scene.Node, zdroj: Sharan (2015, s. 150)

Interakce s uživatelem

Zpracování interakce s uživatelem probíhá podobně jako v knihovně Java Swing s tím rozdílem, že se již nepracuje s pojmem „posluchač“ (Listener), nýbrž „obsluhovačem událostí“ (Event Handler), tj. typované třídy implementující rozhraní *javafx.event.EventHandler<T extends Event>.*

V rámci jednotlivých tříd prvků uživatelského rozhraní je implementována například metoda *Node#setOnMouseClicked()*, prostřednictvím které lze zaregistrovat

metodu pro obsluhu událostí (tj. Event Handler) pro zpracování události kliknutí tlačítka myši. Registraci obsluhovače událostí lze provést i prostřednictvím metody *Node#addEventHandler()*, popřípadě pomocí *Node#addEventFilter()*, pokud bude v roli filtru událostí, viz následující text.

Fundamentálním konceptem je však proces zpracování události v rámci grafu scény. Proces zahrnuje čtyři fáze:

- **fáze odvození cíle** – pro každou událost je nutné odvodit její cíl, přičemž cílem může být libovolná instance třídy *javafx.scene.Node*, např. při stlačení klávesy je cílem grafický prvek, který je v tomto okamžiku označen (focused),
- **fáze konstrukce cesty** – v případě hlubších grafů scény je nutné vypočítat všechny jeho uzly, kterým bude událost předána ke zpracování,
- **fáze zachytávání události** – událost je předávána směrem dolů k listům grafu scény a je dále filtrována dle registrovaných filtrů událostí,
- **fáze probublání události** – prvním místem, kde je zpracována událost pomocí registrovaného obsluhovače událostí je listový uzel, poté událost „probublá“ směrem ke kořenu, přičemž v rámci dvou posledních fází lze událost „zkonzumovat“, tedy zabránit její další propagaci. (Sharan, 2015)

Aplikace vzoru Model-View-Controller

Aplikaci vzoru Model-View-Controller (dále jen MVC) lze ve frameworku JavaFX docílit vázáním (binding), příp. odposloucháváním změn (listening), dat. Na rozdíl od klasického návrhu vzoru MVC v programovacím jazyce Java, tedy na základě třídy *java.util.Observable* a rozhraní *java.util.Observer***,** lze v rámci tohoto frameworku definovat tzv. vlastnosti (properties). Vlastnosti jsou reprezentovány celou kolekcí tříd implementujících rozhraní *javafx.beans.Observable* a mnoha dalšími v závislosti na typu proměnné (řetězec, číslo, pravdivostní hodnota). Ve výsledku jsou data vázána pomocí metody *bind()*, popřípadě lze registrovat posluchače změn – třídu implementující rozhraní *javafx.beans.value.ChangeListener<T>*.

Vícevláknové programování

Jedním z klíčových otázek zajištění bezproblémové uživatelské přívětivosti aplikace je její responsivnost. Z tohoto důvodu není doporučováno vykonávat

dlouhotrvající činnosti pomocí hlavního vlákna aplikace (stejný koncept je zaveden i pro GUI aplikace určené pro OS Android). Operací, pro které je vhodné dedikovat nové programové vlákno, je například práce se souborovým systémem a soubory (ukládání, načítání) nebo síťová komunikace. Aby bylo GUI programu a vázání dat (implementace MVC) deterministické, prvky uživatelského rozhraní a vlastnosti mohou být modifikovány pouze z hlavního programového vlákna dané aplikace.

Jedním z postupů, jak modifikovat GUI aplikace, je použití metody *Platform#runLater()*, jejímž parametrem je třída implementující rozhraní *java.lang.Runnable*. V rámci této třídy (popřípadě lambda výrazu) lze specifikovat libovolnou operaci s GUI, není však zaručeno, že proběhne okamžitě.

3.2.3 Projekt Gluon Mobile

Cílem projektu Gluon Mobile je umožnit vývoj aplikací pro všech pět hlavních desktopových i mobilních platforem ve frameworku JavaFX. (Gluon, c2017a)

Prostřednictvím knihoven projektu lze abstrahovat od hardwaru a využívat systémové zdroje a služby abstraktně – například GPS, bluetooth, uložiště, akcelerometry nebo multidotyková zobrazovací zařízení. (Gluon, c2017a)

Aplikace využívají dostupnou HW akceleraci a grafický výkon zařízení, je tedy možné vyvíjet i aplikace náročnější na výkon, který by nebylo možné zajistit při využití webových prohlížečů nebo nativních kontejnerů (wrapper). (Gluon, c2017a)

JavaFXPorts

Konkrétní součástí projektu Gluon Mobile, jejíž prostřednictvím probíhá překlad a uzpůsobení aplikace implementované prostřednictvím frameworku JavaFX pro jednotlivé cílové platformy, se nazývá JavaFXPorts.

JavaFXPorts je možné využívat jako zásuvný modul v rámci třech nejrozšířenějších integrovaných vývojových prostředí NetBeans, IntelliJ a Eclipse IDE. Prostřednictvím zásuvného modulu lze vygenerovat nový projekt, jehož knihovny jsou spravovány a kompilace je zajišťována programovým systémem Gradle (viz kapitola 4.2.2).

V dalším textu je verze pro operační systém Android označována jako „*Android port*“ a verze pro operační systém iOS jako „*iOS port*“

Princip překladu aplikace pro OS Android

Android port je kompilován tak, že je zdrojový kód aplikace napsaný v programovacím jazyce Java přeložen a poté přizpůsoben cílové platformě Android. Překlad je zajištěn zásuvným modulem nástroje Gradle (viz kapitola 4.2.2). Na zdrojový kód aplikace a využité knihovny se však vztahují omezení uvedená v kapitole 4.5.

Princip překladu aplikace pro iOS

V porovnání s principem překladu pro cílovou platformu Android je v případě *iOS portu* nutno zdrojový kód se všemi využitými knihovnami přeložit z programovacího jazyka Java přímo do strojového kódu určeného pro cílovou platformu iOS. Tento proces je zajištěn AOT kompilátorem *RoboVM*. Jedná se však o složitý a časově náročný proces, v jehož rámci je nutné zkompilovat zdrojový kód aplikace včetně všech využitých knihoven dopředu, ještě před spuštěním aplikace. Zkušenosti autora této práce s překladem pro platformu iOS jsou uvedeny v kapitole 4.5.5.

3.3 Cílové platformy

Cílovými platformami (resp. operačními systémy) vyvíjené aplikace jsou následující:

- Microsoft Windows,
- macOS,
- Linux,
- Google Android,
- Apple iOS.

Jednotlivé platformy jsou v dalších odstavcích blíže rozebrány z pohledu jejich základní architektury, typu podporovaných zařízení, stručné historie, nutnosti optimalizace vyvíjené aplikace pro danou platformu a v neposlední řadě tržního podílu jejich verzí. Vždy je také uvedena nejnovější verze daného operačního systému.

3.3.1 MS Windows

Operační systém MS Windows společnosti Microsoft se řadí mezi nejrozšířenější operační systémy na světě s více než třicetiletou historií. První verze tohoto operačního systému – MS Windows 1 – byla vydána v roce 1985 s a na rozdíl

od svého předchůdce, tj. operačního systému MS-DOS, je MS Windows víceúlohovým operačním systémem. (Gibbs, 2014)

Následující text bude věnován jeho nejnovější verzi, která je vyvíjenou aplikací plně podporována, není však vyloučena podpora předchozích verzí, kde záleží na kompatibilitě JRE (Java Runtime Environment).

Windows 10

Nejnovější verze operačního systému, společnosti Microsoft, MS Windows 10, je určená jak pro desktopové PC (architektury x86), tak i pro mobilní zařízení, tzv. Tablet PC. Hlavním rozdílem Tablet PC od desktopového PC je to, že podpora multidotykových zobrazovacích zařízení je samozřejmostí. Tablet PC lze přitom velikostí přirovnat k netbooku. Vyvíjená aplikace podporuje i verzi pro desktopová multidotyková zařízení a bude tedy využitelná i na Tablet PC.

Nutnost optimalizace vyvíjené aplikace

Aplikaci není potřeba optimalizovat, jediným požadavkem je instalace běhového prostředí JRE verze 8u40 (2015). Testování aplikace probíhalo na počítači s 64-bitovým operačním systémem MS Windows 10 s monitorem o rozlišení 1920x1080.

3.3.2 macOS

macOS, spravovaný společností Apple, je unixově založený operační systém pro desktopová zařízení Mac. Přesněji jde o zařízení typu přenosného počítače iMac a typu notebooku MacBook.

Operační systémy macOS byly v minulosti označovány Mac OS, Mac OS X, OS X a nyní od roku 2016 macOS. Historie vývoje operačního systému macOS začíná v 80. letech vydáním přenosného počítače s GUI s operačním systémem NeXTSTEP. První verzí Mac OS X je Mac OS X 10.0 s kódovým označením Cheetach z roku 2001. V současné době je k dispozici verze macOS Sierra (10.12 z pohledu verzování Mac OS X). (Grebeň, 2016)

Stejně jako v případě operačního systému MS Windows se kompatibilita aplikace odvíjí od dostupnosti JVM pro danou verzi operačního systému macOS.

Nutnost optimalizace vyvíjené aplikace

Aplikaci není potřeba optimalizovat, jediným požadavkem je instalace běhového prostředí JRE verze 8u40 (2015). JRE musí pocházet z distribuce společnosti Oracle, nelze ji využít v kombinaci s OpenJDK.

3.3.3 Linux

GNU/Linux, nebo jen Linux, je svobodný operační systém. Uživatelé jej mohou instalovat na libovolný hardware a používat k soukromým i komerčním účelům. Autory GNU/Linux jsou Richard Stallman a Linus Torvalds, který vytvořil jádro tohoto operačního systému (1984). (Root.cz, c1998–2017)

Operační systém Linux jako celek nemá žádné oficiální verze jako většina operačních systémů. Skládá se z jádra (kernel), v současné době ve stabilní verzi 4.10.2 z 12. 3. 2017, a dodatečných balíčků, konfiguračních nástrojů a dalších komponent, které dle Root.cz (c1998–2017) tvoří dohromady tzv. distribuci.

K dispozici jsou desítky distribucí operačního systému Linux. Zatímco některé jsou přizpůsobeny k využití jako operační systémy aplikačních serverů (například CentOS), distribuce Ubuntu nebo SUSE lze využívat jako operační systémy osobních počítačů, notebooku a jiných zařízení.

Nutnost optimalizace vyvíjené aplikace

Aplikaci není potřeba optimalizovat, jediným požadavkem je instalace běhového prostředí JRE verze 8u40 (2015). JRE musí stejně jako v případě macOS pocházet z distribuce společnosti Oracle.

3.3.4 Google Android

Google Android, v roce 2016 nejrozšířenější mobilní operační systém na světě (tržní podíl 79,6% dle IDC, c2017), lze využívat v zařízeních všech kategorií (chytré telefony nebo hodinky, tablety, televize a další), a to jak verzi pro architekturu x86, tak ARM (Advanced RISC Machine). (Smith, 2015)

Počátky tohoto operačního systému jsou datovány přibližně do roku 2006, kdy vznikaly první prototypy běžící na mobilním telefonu T-Mobile G1. Na vývoji operačního systému Android tehdy pracovali vývojáři společnosti Android Inc., která byla následně ze strategických důvodů koupena společností Google Inc. s cílem vlastnit společnost pro vývoj software určeného pro chytré telefony, neboť

v následujících letech byl očekáván boom chytrých zařízení také v reakci na plány společnosti Apple. (Dobbie et al., c2016) (Smith, 2015)

V rámci vývoje aplikace pro operační systém Android je nutné stanovit, jaké verze budou aplikací podporovány, liší se totiž možnostmi vývoje (například prvky nebo princip zobrazování uživatelského rozhraní), podporovanými systémovými službami a v neposlední řadě lze odhadnout i výkon zařízení – chytré zařízení s Androidem 2.1 zřejmě nebude vybaveno více než 512 MB operační paměti a více než dvoujádrovým procesorem, zatímco v případě zařízení s Androidem 6.0 může jít i o 4 GB operační paměti s osmi jádrovým procesorem.

V tabulce 3-1, převzaté z Android Developers (c2017), jsou uvedeny aktuální statistiky tržních podílů jednotlivých verzí operačního systému Android.

Tabulka 3-1: Tržní podíl verzí OS Android, zdroj: Android Developers (c2017), upraveno

Verze	Název	Podíl na trhu
2.3.3 - 4.0.4	Gingerbread - Ice Cream Sandwich	1,8%
4.1 - 4.3	Jelly Bean	10,1%
4.4	Kitkat	20,0%
5.0 - 5.1	Lollipop	32,0%
6.0	Marshmallow	31,2%
7.0	Nougat	4,9%

Nutnost optimalizace vyvíjené aplikace

Pro Google Android bylo zpracováno nové grafické uživatelské rozhraní, které bude přizpůsobeno velikosti zobrazovacích zařízení a také rozdílné interakci s uživatelem (vstupním zařízením není myš, ale události dotyků zobrazovacího zařízení). Dále je vyžadován **tablet s Androidem 4.4 a vyšším**, tj. podpora zařízení s obrazovkou menší než 7 palců vyžaduje další úroveň optimalizace.

Dalším prvkem UX (uživatelský prožitek) je práce s úložištěm mobilních zařízení, které bude blíže popsáno v kapitole 4.7.2.

Aplikace byla testována na Tabletu Lenovo Tab3 8 s operačním systémem Android 6.0 o velikosti obrazovky 8 palců s rozlišením 1280x800 pixelů.

Prostřednictvím zásuvného modulu JavaFXPorts lze zkompilovat aplikaci jen pro ARM architekturu (v dokumentaci se nachází tato volba jen pro iOS port), což výrazným způsobem ztěžuje testování na zařízení architektury x86.

3.3.5 Apple iOS

Operační systém iOS je vyvíjen od roku 2008 (tehdy jako iPhone Firmware) společností Apple exkluzivně pro zařízení značky Apple, a to iPhone, iPod touch a iPad. Jedná se tedy o výhradně mobilní operační systém ARM, s pozicí po Androidu druhého nejrozšířenějšího operačního systému na světě v roce 2016 (tržní podíl 18,7% dle IDC: Smartphone OS Market Share 2016, c2017). K dnešnímu datu (3. 4. 2017) je k dispozici verze iOS 10.3. (Apple Developer, 2017)

Zařízení s rozdílnou verzí operačního systému iOS se stejně jako v případě OS Android liší rozměry, výkonem ale také funkcionalitou, kterou může vývojář aplikace využít. Je tedy vhodné uvést základní statistiky využití jednotlivých verzí iOS dle Apple Developer (2017) (viz tabulka 3-2).

Tabulka 3-2: Podíl na trhu jednotlivých verzí iOS, zdroj: Apple Developer (2017), upraveno

Verze	Podíl na trhu v %
Starší	5 %
9	16 %
10	79 %

Výčet podporovaných verzi iOS nebyl nalezen v rámci dokumentace Gluon Mobile, autoři tohoto projektu však zaručují kompatibilitu s nejnovější verzí iOS 10.

Nutnost optimalizace vyvíjené aplikace

Verze pro Apple iOS je totožná s verzí pro Google Android a je také určena výhradně pro zařízení typu tablet (iPad).

4 Návrh a implementace aplikace pro vizualizaci a simulaci procesních Petriho sítí

Předmětem této kapitoly je provést čtenáře procesem návrhu a implementace multiplatformní aplikace umožňující návrh, simulaci a verifikaci procesních P/T Petriho sítí a následně i obeznámit s uživatelskými rozhraním a funkcemi výsledného programového řešení.

4.1 Nástroje a metodiky návrhu a implementace aplikace

V této kapitole jsou definovány nástroje a metodiky, na které je v dalším textu odkazováno, rovněž bylo nutné zavést další základní pojmy. Jedná se tedy o teoretický úvod do problematiky návrhu a implementace aplikace.

4.1.1 UML

I přestože je UML (unifikovaný modelovací jazyk) od roku 1997, kdy byl přijat organizací OMG (Object Management Group), standardizovaným grafickým nástrojem používaným primárně jako grafická notace v rámci metodik UP (Unified Process) a RUP (Rational Unified Process), lze jej využívat i samostatně pro zpracování objektově orientované analýzy (viz odstavec níže). Účelem UML je umožnit návrhářům softwaru graficky zpracovat strukturální a behaviorální složku vyvíjeného produktu tak, aby šlo výsledné artefakty (diagramy, tabulky, popisy, scénáře) využít jako forma komunikace v týmu vývojářů a zároveň i v komunikaci se zákazníky, resp. zadavateli, a to hlavně v případě diagramů o vysoké úrovni abstrakce, např. diagram případů užití. (Hamilton a Miles, 2006) (Čechák, 2015)

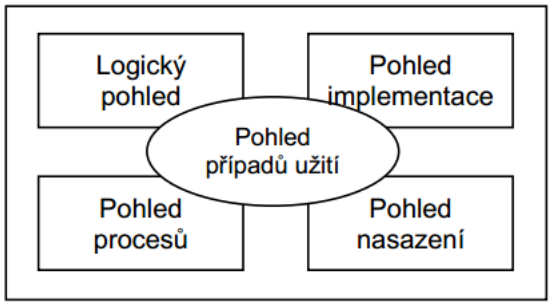
Objektově orientovaná analýza

Grafická notace UML slouží hlavně pro zpracování objektově orientované analýzy. Objektově orientovaná analýza je založena na pojmu třídy – speciálním datovém typu, v jehož rámci lze deklarovat a definovat jak data (datové položky), tak funkcionalitu (funkce, procedury, metody).

Model pohledů 4+1 a diagramy

Autorem modelu pohledů 4+1 (viz obr. 4-1) je Philip Kruchten. Modelem se rozumí zařazení nástrojů UML z pěti různých pohledů do pěti následujících skupin:

- **logický pohled** – abstraktní struktura systému, souvisí s diagramem tříd, diagramy interakce, diagramem objektů a stavového automatu,
- **pohled procesů** – diagram aktivit, na jehož základě jsou definovány procesy v systému,
- **pohled nasazení** – komponenty znázorňující fyzické nasazení systému, například diagram nasazení,
- **pohled implementace** – jedná se o popis architektury systému prostřednictvím diagramů komponent a balíčků. (Arlow a Neustadt, 2007) (Čechák, 2015)



Obrázek 4-1: Diagram 4+1 pohledů, zdroj: Čechák (2015, s. 31)

Všechny výše zmíněné pohledy jsou zastřešeny **pohledem případů užití**, ze kterého je modelována funkcionalita systému za využití diagramů případu užití a doprovodné dokumentace. (Arlow a Neustadt, 2007)

V rámci standardu UML je definováno více než 15 diagramů. Pro účely návrhu a implementace vyvíjené aplikace byly zpracovány následující diagramy:

- diagram tříd,
- diagram komponent a balíčků,
- diagram aktivit,
- sekvenční diagram,
- diagram objektů,
- stavový diagram.

S jednotlivými diagramy bude čtenář obeznámen v rámci druhé částí kapitoly 4.

4.1.2 Projektová metodika Kanban

Kanban patří mezi agilní projektové metodiky, které kladou vysoký důraz na grafické předání informace. Dle Janiš (2013) jde spíš o sadu principů regulujících workflow projektu a přípravu na aplikaci projektové metodiky Scrum. Dle Radigan

(c2017) jde o více než 50 let starou metodiku, populární hlavně mezi týmy v oblasti agilního vývoje softwaru.

Kanban pochází z Japonska, kde byl mimo jiné aplikován v řízení výroby ve spojitostí se systémem tahu. Potřebné součástky byly dodávány až když byla potřeba (just in time). (Šochová a Kunce, 2014)

Hlavním rozdílem Kanbanu v porovnání s ostatními projektovými metodikami je to, že v jeho rámci není nic striktně definováno. (Šochová a Kunce, 2014)

Agilní metody

Slovo agilní lze dle Šochová a Kunce (2014) chápat jako „*dynamický, rychlý, interaktivní, zábavný, hravý, rychle reagující na změnu*“. Jde o jiný způsob života upřednostňující reálný výsledek před striktními procesy a změnu před předem naplánovaným. Agilním chování je hlavně o spolupráci, komunikaci a připravenosti na změnu. Agilní projekt není řízen striktně, nicméně není to chaos – má jasná pravidla. (Šochová a Kunce, 2014)

Vývoj aplikace je zastřešen projektovou metodikou Kanban, kvůli jednoduchosti, srozumitelnosti a charakteru projektu, který se vyvíjel kontinuálně bez iterací.

Šest základních principů Kanbanu

Dle Smartsheet (c2017) lze definovat šest základních principů Kanbanu, z nichž tři jsou uvedeny v následujících odstavcích.

Vizualizovat svojí práci

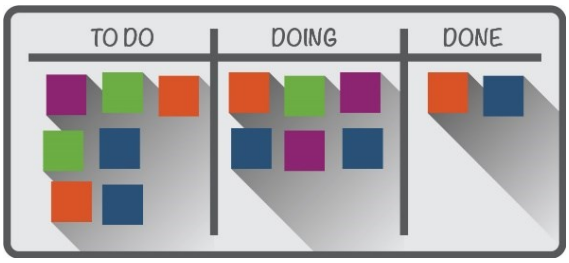
Již v úvodu kapitoly bylo uvedeno, že je kladen důraz na grafické předání informace. Za tímto účelem se v Kanbanu využívají tabule, ze kterých jednotliví členové týmu mohou snadno vyčíst, na čem se pracuje a kdo na dané činnosti pracuje.

Úkoly (nebo také činnosti) jsou graficky zpracovány ve formě barevných kartiček. Nejzákladnější Kanban tabule (viz obr. 4-2) obsahuje tři následující sloupce:

- úkoly ke zpracování (*TO DO*) – úkoly jsou obvykle založeny na uživatelských příbězích (*user stories*), které se nachází v seznamu zvaném *Backlog*,

- úkoly na kterých se pracuje (*Work in Progress*),
- dokončené úkoly (*Done*).

Jednotlivé dokončené úkoly nemusí být úspěšně dokončené, ale i například zrušené nebo uzavřené s tím, že jednou budou dokončeny.



Obrázek 4-2: Kanban tabule, zdroj: Smartsheet (c2017)

Dle složitosti projektu lze tabuli rozšířit, nebo pozměnit smysl sloupců, například, dle metodiky UP na sloupce – *modelování procesů, analýza, návrh, implementace, testováni a dokončeno*.

Omezovat rozpracované činnosti

Jednotliví členové týmu by neměli mít rozpracované vyšší množství úkolů. Na druhou stranu však nelze stanovit jednotný počet rozpracovaných úkolů pro všechny projekty a jejich projektové fáze.

Spravovat tok práce

Výhodou metodiky je to, že se všechny procesy nemusí zavádět okamžitě (nebo může sloužit také jako způsob přechodu na složitější metodiky). Členové týmu by se měli učit z chyb, postupně zpřesňovat odhady a také rychle odhalit úzká místa.

Rozšíření principu Kanbanu

Obvykle je čistý princip Kanbanu ještě doplněn například následujícím:

- pravidelnou retrospektivou,
- mítinky vestoje (standup meetings),
- XP (Extreme Programming) praktikami – jako je párové programování, kontinuální integrace, kolektivní zodpovědnost nebo jednoduchý design spojený s refaktoringem,
- definice rolí v týmu. (Šochová a Kunce, 2014)

Rozdíly mezi projektovou metodikou Scrum a Kanbanem

Tyto metodiky mohou být srovnány například dle Radigan (c2017). (viz tab. 4-1).

Tabulka 4-1: Srovnání metodiky Scrum a Kanban, zdroj: Radigan (c2017), upraveno

	SCRUM	KANBAN
Kadence	Sprinty s pevně danou dobou trvání	Kontinuální tok
Metodika vydávání verzí softwaru	Na konci každého sprintu po schválení vlastníka produktu	Kontinuální vydávání nových verzí
Role	Vlastník produktu, Scrum master, tým vývojářů	Žádné pevně stanovené role
Klíčové metriky	Rychlost	Délka cyklu
Změna filozofie	V rámci jednoho sprintu by již nemělo docházet ke změnám plánu daného sprintu	Ke změně může dojít kdykoliv

4.2 Nástroje správy softwarových projektů

V rámci každého softwarového projektu je nutné zabezpečit následující procesy:

- správa zdrojového kódu (uložení, zabezpečení, dostupnost),
- správa vlastních knihoven,
- správa využitých knihoven třetích stran,
- princip kompilace a programových systémů,
- princip testování,
- dostupnost dokumentace.

Tyto procesy lze řídit ručně, ale v praxi se vyplatí využít jeden z nástrojů, který procesy poté zabezpečuje a lze je lehce konfigurovat. Pro správu vyvíjené aplikace byly využity nástroje Apache Maven a Gradle, popsané v následujících kapitolách.

4.2.1 Apache Maven

Apache Maven je nástrojem správy softwarových projektů založený na konceptu projektově objektového modelu (POM). Pomocí tohoto konfiguračního souboru lze efektivně z jednoho místa řídit veškeré výše uvedené procesy.

Apache Maven je hlavním nástrojem správy vyvíjené aplikace.

Historie

Apache Maven vznikal v prostředí projektu Jakarta Alexandria, ze kterého byl vyčleněn v roce 2001. Jde tedy o jeden z nejstarších v současné době stále využívaných nástrojů pro správu softwarových projektů. (Zyl, 2017)

Oblasti využití

Nástroj je využíván hlavně v oblasti vývoje v programovacím jazyce Java, v oblasti vývoje mobilních aplikací však byl nahrazen novějšími nástroji, například nástrojem Gradle. Díky zásuvným modulům jsou podporovány i ostatní programovací jazyky, jako je např. Scala nebo C/C++.

Projektový objektový model (POM)

Projektový objektový model představuje textovou (XML) reprezentaci projektu z pohledů konfigurace, programových knihoven, programových modulů, principů kompilace, zásuvných modulů, testovaní a dalších. Struktura POM dokumentu zde nebude kvůli svému rozsahu rozebrána.

Programový modul

Programový modul (v případě programovacího jazyka Java) je množina programových jednotek (tříd, rozhraní, výčtů) seskupených v balíčcích a prostředků (resources – obrázky, soubory, properties soubory a další). Každý programový modul je definován pomocí vlastního projektového objektového modelu, v jehož rámci lze definovat hierarchie rodič-potomek mezi více programovými moduly. Koncepce kapitoly č. 4.3 je založena na modulární struktuře vyvíjené aplikace, jde právě o výše popsané moduly.

Repositáře

Programové knihovny projektu jsou pomocí tohoto nástroje stahovány do zařízení vývojářů z tzv. *vzdálených repositářů*, tj. vzdálených úložišť umístěných v internetu. Diskový prostor, kam jsou knihovny staženy, se nazývá *lokální repositář*. Na základě tohoto principu není potřeba knihovny fyzicky spravovat, ale je možné je lehce využít v projektu.

4.2.2 Gradle

Moderní nástroj, který je dle jeho autorů výrazným skokem ve správě softwarových projektů v prostředí JVM. Je založen na projektu Apache Ivy (nástroj pro správu programových knihoven). Je kompatibilní s repositáři určenými pro nástroje Apache Maven a Apache Ivy a v neposlední řadě umožňuje popsat projektovou strukturu bez použití XML, a to jazykem Groovy, což tohle z něj dělá vysoce flexibilní nástroj. (Gradle Inc., c2017)

V rámci diplomové práce jde o podpůrný nástroj určený pro kompilaci mobilní verze aplikace, programové moduly budou konfigurovány pomocí obou nástrojů.

Oblast využití

Gradle není vázán na jazyk zdrojového kódu projektu. Je však oficiálním nástrojem pro vývoj aplikací pro platformu Android v prostředí integrovaného vývojového prostředí Android Studio.

4.3 Koncept a základní požadavky na aplikaci

Koncept vychází z toho, že se má jednat o multiplatformní aplikaci, což byl také první nefunkční požadavek, známý již před začátkem vývoje. Prvním, a zřejmě také klíčovým, krokem byl výběr technologie.

4.3.1 Výběr technologie

Již v třetí kapitole byl popsán programovací jazyk Java, framework JavaFX a projekt Gluon Mobile. Tato kombinace prostředků pak byla využita i v rámci návrhu a samotné implementace aplikace.

Z ostatních zvažovaných technologií šlo například o následující:

- **Qt –** framework umožňující vývoj multiplatformních aplikací hlavně v jazycích C/C++,
- **Delphi 11 Embarcadero** – framework založený na jazyku Object Pascal,
- **Apache Cordova** – projekt multiplatformního vývoje ve skriptovacím jazyce Java Script založený na principu HTML Rendering Engine, kdy aplikace běží v programovém kontejneru, který ji vykresluje jako dynamickou webovou stránku.

Framework JavaFX ve spojení s projektem Gluon Mobile byly vybrány mimo jiné z následujících důvodů:

- **zaměření autora této práce** (webové aplikace v Javě),
- **rozšířenosti programovacího jazyka Java** (na rozdíl např. od Delphi),
- **portabilitě samotné platformy** (díky JVM),
- **možnostem frameworku JavaFX** v oblastech:
  - návrhu uživatelského grafického rozhraní,
  - zpracování událostí,
  - vykreslování geometrických útvarů (Shape API).

4.3.2 Požadavky

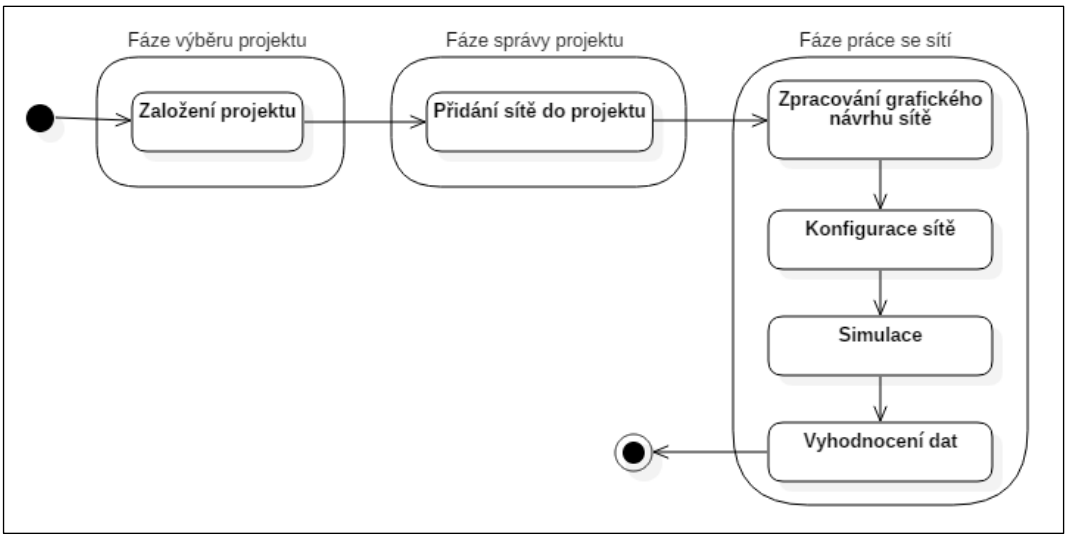
Vývoj aplikace je metodicky zastřešen, jak bylo uvedeno již v kapitole 4.1.2, projektovou metodikou Kanban. Požadavky tedy byly specifikovány kontinuálně v průběhu vývoje a bez jakýchkoliv iterací. Za zmínku však stojí požadavky, stanovené již v rámci výběru technologie (viz tab. 4-2).

Tabulka 4-2: Požadavky známé před začátkem vývoje, zdroj: vlastní

Id	Název a popis požadavku	Kategorie
1	Podpora desktopových i mobilních platforem (viz kapitola 3.3)	Nefunkční
2	Grafický editor P/T Petriho sítí	Funkční
3	Možnost simulace P/T Petriho sítí	Funkční
4	Jedno GUI pro desktopové platformy, které lze přizpůsobit mobilním platformám (UX)	Funkční
5	Příprava aplikace pro implementaci více tříd Petriho sítí, popřípadě vlastních tříd vzniklých v rámci výzkumné činnosti vědeckého týmu vedoucího této práce	Obchodní

4.3.3 Uživatelský příběh

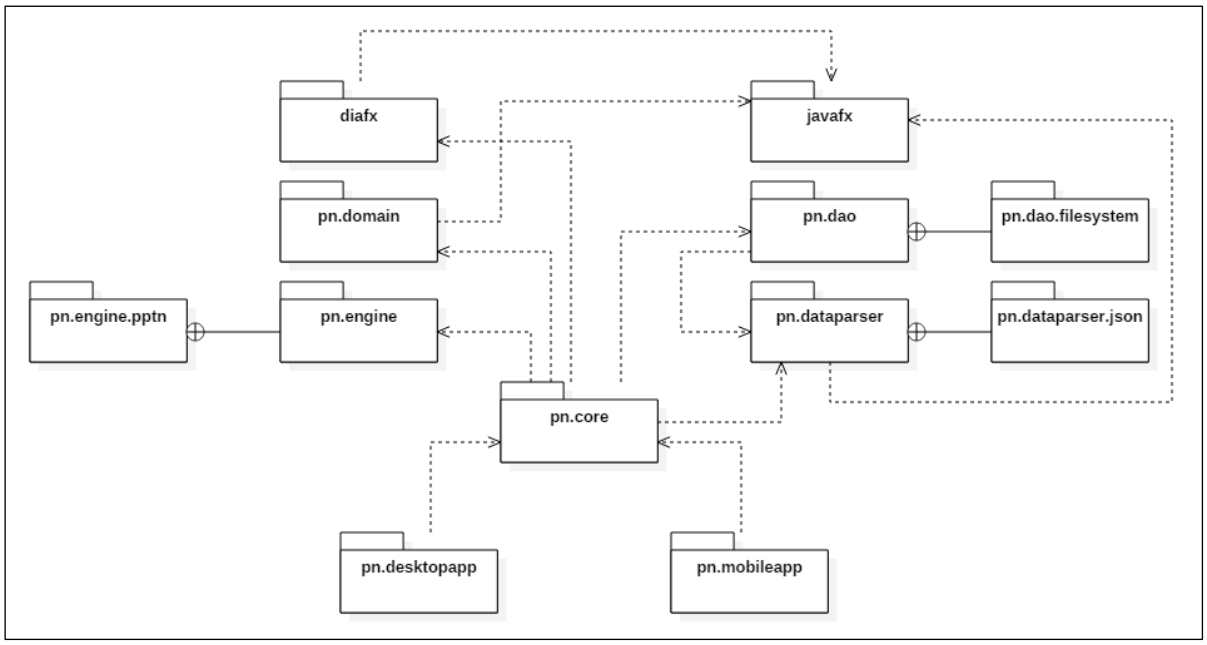
Na základě počátečních požadavků byl vzhledem k cílovým platformám a jejich možnostem stanoven základní způsob využití aplikace, který je zpracován pomocí diagramu aktivit (viz obr. 4-3). Tento způsob byl později nazván jako „*koncepce workflow aplikace*“ a je představen zároveň s výslednou aplikací v kapitole 4.6.



Obrázek 4-3: Plánovaný způsob práce s aplikací, zdroj: vlastní

4.4 Modulární návrh aplikace

Vyvíjená aplikace má silně modulární charakter, nejde však o moduly ve formě samostatně spustitelných programových systémů. Moduly jsou reprezentovány programovými moduly nástroje pro správu softwarových projektů Maven (viz obr. 4-4).



Obrázek 4-4: Architektura aplikace, zdroj: vlastní

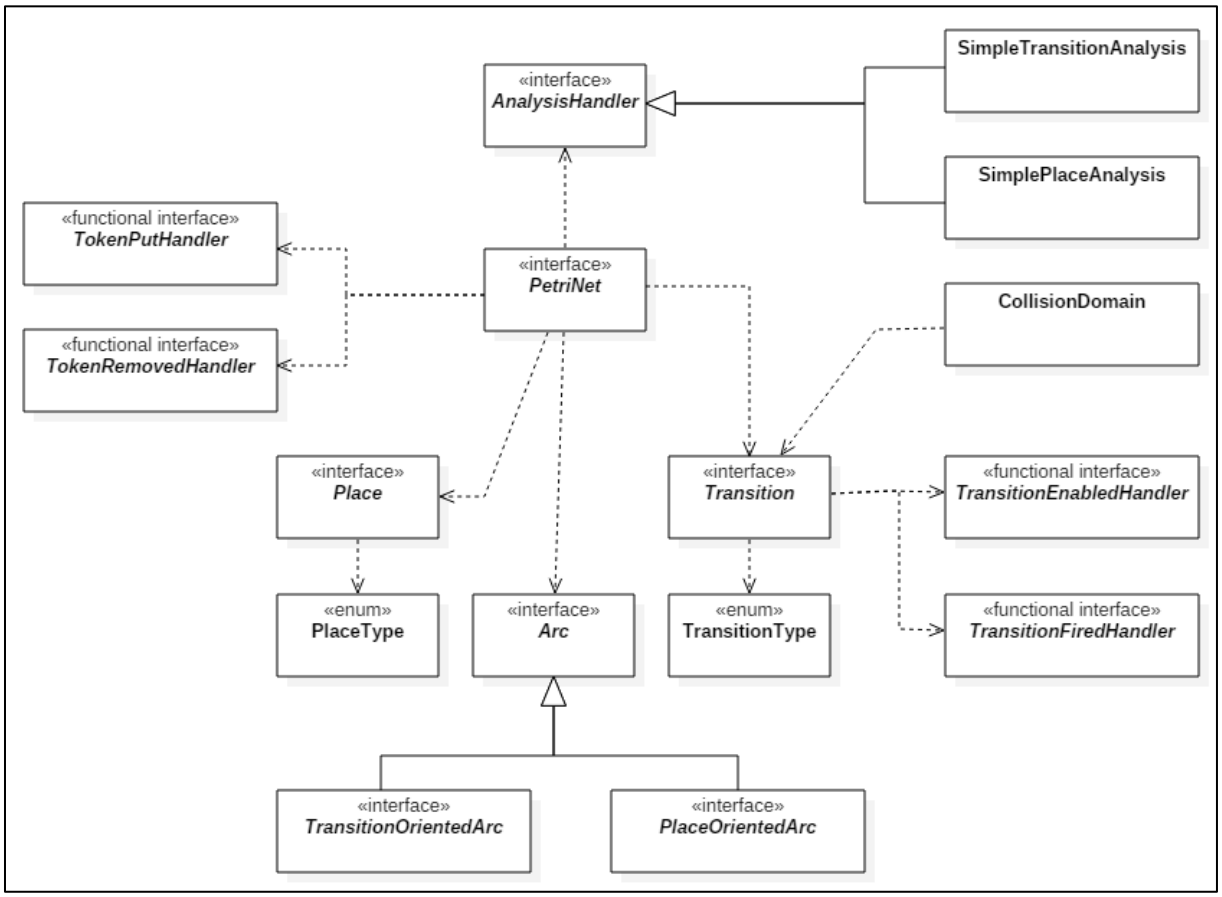
Jednotlivé moduly jsou sdruženy pomocí POM projektu, zastupujícího roli rodičovského projektu (*Parent POM Project*).

UML dokumentace byla zpracována v programu StarUML 2. V dokumentaci je abstrahováno od detailu návrhu, pokud není uvedeno jinak.

4.4.1 Moduly logiky Petriho sítí

Modul logiky Petriho sítí (dále jen *PN Engine*) byl navržen a vyvíjen jako první. Celá aplikace tvoří MVC (Model-View-Controller) nadstavbu právě tohoto modulu. Logikou Petriho sítí je myšlen hlavně jejich model (komponenty, viz kapitola 2.5 a 2. 8), analýza stavových a běhových vlastností, validace sítě a v neposlední řadě i průběh simulace. Modul zahrnuje i generátor grafu dosažitelných značení (viz kapitola 2.7.4).

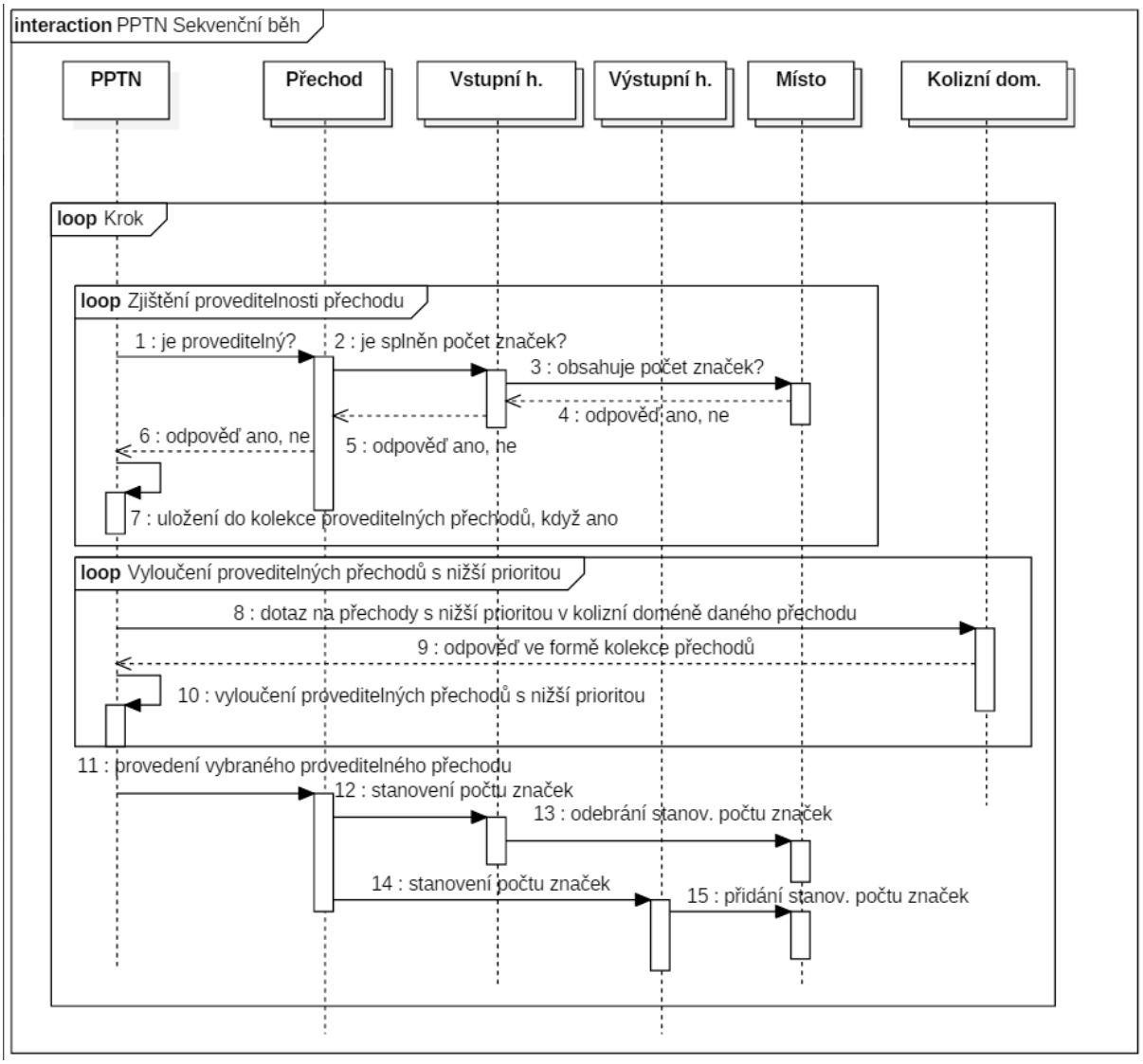
Návrh modelu Petriho sítě (dále jen PN) je postaven na klasických OOP konceptech **dědičnosti, zapouzdřenosti a mnohotvárnosti**. Jednotlivé komponenty PN jsou reprezentovány třídami, které implementují odpovídající rozhraní (*Place, Transition*), a to proto aby mohly vznikat hierarchie implementačních tříd. Pro samotnou PN je určeno rozhraní *PetriNet*, jehož implementací je například třída *ProcessPTPetriNet*. Celá hierarchie tříd je zobrazena na obrázku 4-5 prostřednictvím diagramu tříd.



Obrázek 4-5: Diagram tříd modulu pn-engine, zdroj: vlastní

Zatímco v případě modulu *PN Engine* (*pn-engine*) jde hlavně o abstraktní návrh (vyjma univerzálních knihoven pro analýzu a validaci), implementací je v současné době modul *PPTN Engine* (*pn-engine-pptn*). Tento modul reprezentuje konkrétní implementaci typu Petriho sítí, a to **procesní P/T Petriho sítě**.

Simulace probíhá na principu OOP. Pomocí třídy *ProcessPTPetriNet* je korigován sekvenční běh (viz obr. 4-6), kdy se volají metody přechodů. Díky vlastnostem zapouzdřenosti a mnohotvárnosti lze pak snadno modifikovat jejich implementaci. V rámci simulace dané Petriho sítí je pak nejprve zjištěna proveditelnost jednotlivých přechodů a následně je jeden z nich vybrán k provedení na základě hodnot funkce priority přechodů, které jsou porovnávány v rámci kolizních domén (*CollisionDomain*).

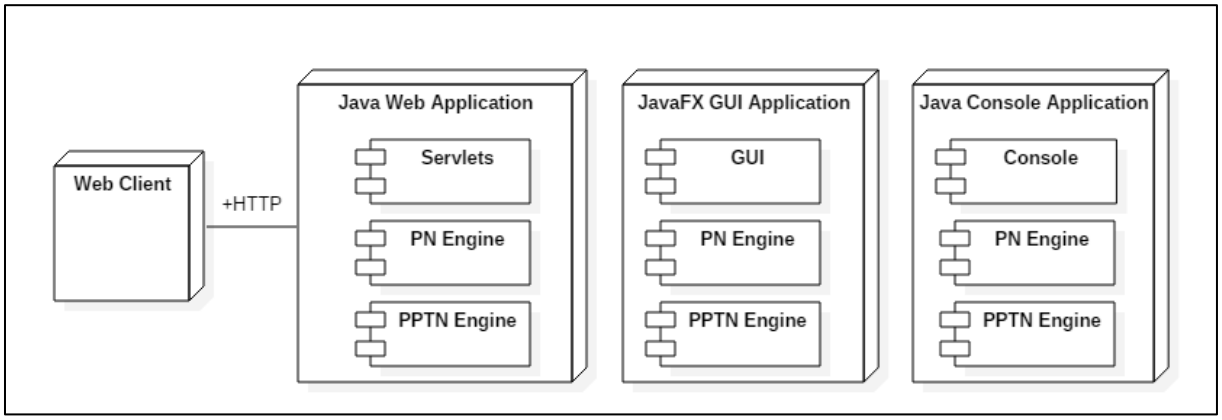


Obrázek 4-6: Sekvenční diagram simulace sekvenčního běhu, zdroj: vlastní

Proces je zachycen pomocí sekvenčního diagramu, kde je však pro lepší pochopení abstrahováno od skutečné implementace a podmínek pro ukončení běhu, tj. množina proveditelných přechodů je prázdná.

Z diagramu je také zjevné, že je návrh připraven např. na dynamické vyhodnocování podmínek jednotlivých hran, protože komunikace mezi přechody a místy probíhá právě jejich prostřednictvím.

Modul je zcela nezávislý na frameworku JavaFX a obsahuje veškeré žádoucí prostředky pro definici, simulaci, konfiguraci, analýzu a validaci procesních P/T Petriho sítí. Neobsahuje však GUI, na druhou stranu jej lze využit již v kombinaci s jednoduchou konzolovou aplikací, která by zajišťovala interakci s uživatelem. Další možností využití tohoto modulu je jeho začlenění do backendu webové aplikace založené na platformě podporující programovací jazyk Java (viz obrázek 4-7).

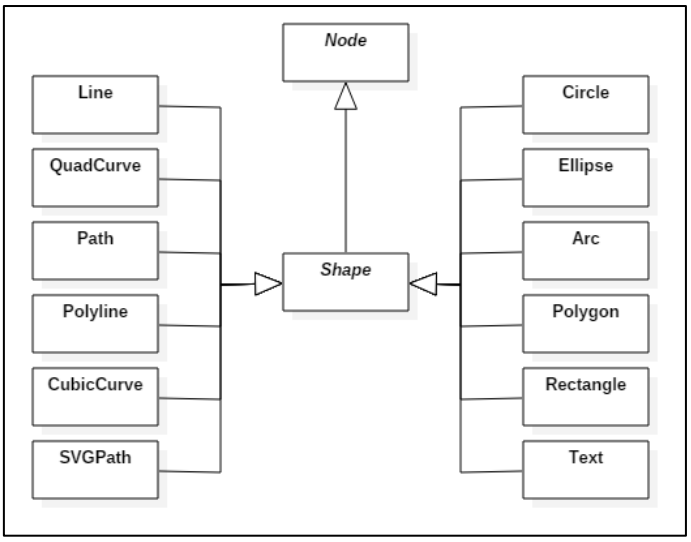


Obrázek 4-7: Diagram nasazení modulu pn-engine, zdroj: vlastní

Vyvíjená aplikace je, jak bylo uvedeno výše, na tomto modulu závislá. Modul logiky Petriho sítí je využíván konkrétně modulem jádra aplikace (viz kapitola 4.4.4).

4.4.2 Modul vykreslování diagramů

Na základě počátečních požadavků, konkrétně požadavku číslo 2 „*Grafický editor P/T Petriho sítí*“, byl jako druhý navržen modul vykreslování diagramů (*DiaFX*), který zahrnuje funkcionalitu vykreslování, chování a interakce s většinou geometrických útvarů (kruhy, čtverce, obdélníky, úsečky atd.). Tato funkcionalita je založena na knihovně *JavaFX Shape API* – knihovně věnované vykreslování, definici a matematickým výpočtům v oblasti analytické geometrie (viz obr. 4-8).

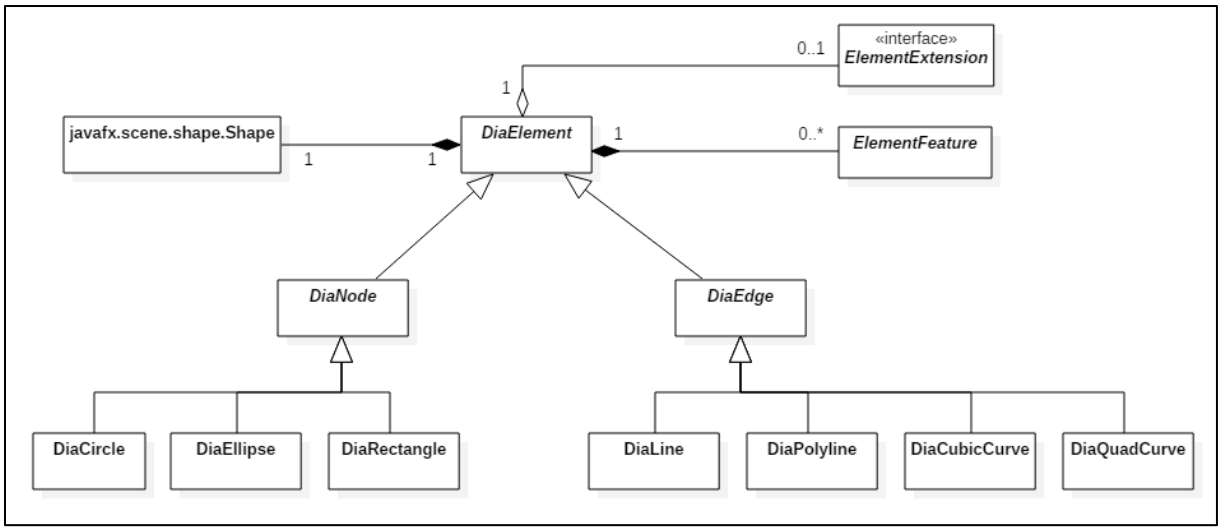


Obrázek 4-8: Hierarchie tříd javafx.scene.shape.Shape, zdroj: vlastní

V rámci modulu vykreslování diagramů však nejde jen o využití *Shape API*, komponenty této knihovny jsou zapouzdřeny třídami modulu *DiaFX net.cemartinapps.diafx.DiaElement*, *net.cemartinapps.diafx.DiaNode*, *net.cemartinapps.diafx.DiaEdge* a jejich implementacemi – ty umožňují, oproti základní funkcionalitě, bohaté možnosti v oblasti napojování jednotlivých uzlů pomocí hran (viz kapitola 4.9.1) a také jejich interakci za pomocí vstupních zařízení jako je myš, ale i dotyková obrazovka. Právě pro práci s dotykovou obrazovkou je celý modul navíc optimalizován a interakce s uživatelem pak probíhá tak, že je aktivní buď režim myši, nebo režim dotyků. Vykreslování je však zajištěno třídami z hierarchie *Shape*.

Výše uvedené je také důvodem, proč nebyla pro vykreslení modelované Petriho sítě využita žádná knihovna třetích stran. Mezi další důvody patří i princip rozšiřování funkcionality jednotlivých komponent modulu. Bylo totiž upuštěno od dědičnosti, ve prospěch kompozice (princip *Composition over Inheritance*). Kompozice je zde řešena pomocí rozhraní *net.cemartinapps.diafx.ElementExtension*.

Pro pochopení nově definovaného API je vhodné prostudovat jeho diagram tříd (viz obr. 4-9).



Obrázek 4-9: Diagram tříd modulu diafx, zdroj: vlastní

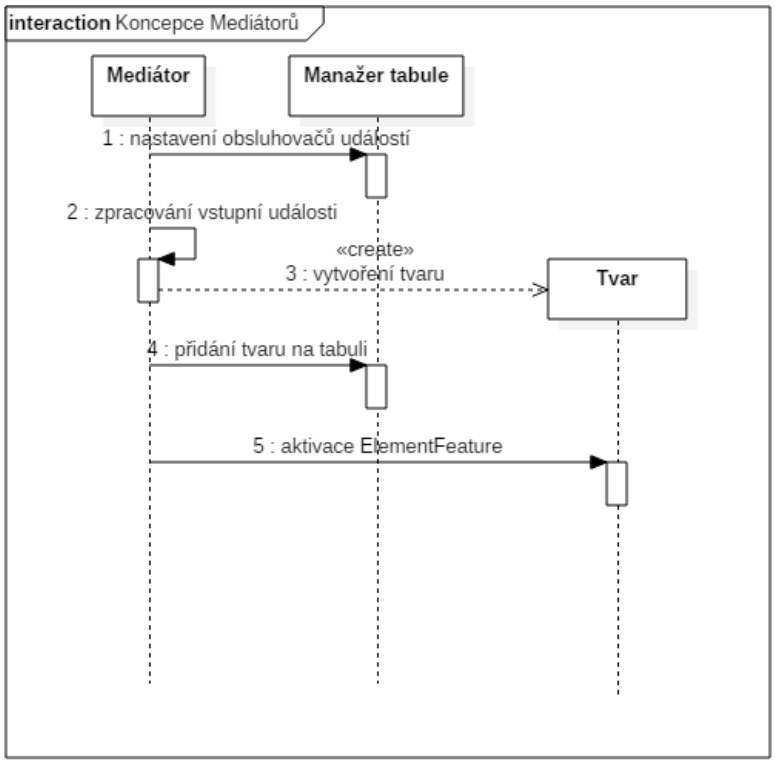
Správa jednotlivých komponent reprezentujících geometrické tvary je zajišťována instancí třídy *net.cemartinapps.diafx.board.BoardManager***.** Komponentu *BoardManager* (dále i manažer tabule) lze propojit s dvojící programátorem dodaných instancí třídy frameworku JavaFX – *Pane* a *ScrollPane*. Prostřednictvím manažera tabule lze poté spravovat prvky umístěné na „*kreslicí tabuli*“ (*Pane*) a jejím rodiči (*ScrollPane*), a tím využít opakovaně celé nově definované API i mimo rámec vyvíjené aplikace, za předpokladu přítomnosti knihovny JavaFX.

Interakce s uživatelem

Z pohledu požadavků na kreslení diagramů bylo nutné navrhnout a implementovat škálovatelný a flexibilní způsob zpracování událostí vstupních zařízení, neboť standardní možnosti frameworku nestačily z hlediska škálovatelnosti chování jednotlivých komponent. Z tohoto důvodu jsou třídami z hierarchie *DiaElement* komponovány implementace abstraktní třídy *net.cemartinapps.features. ElementFeature* **–** jedná se o třídy určené pro zapouzdření správy obsluhovačů událostí. Původně se nazývaly manažery obsluhovačů událostí, ale pro přehlednost byly přejmenovány a jejich současný název lze přeložit jako „rys“ nebo „vlastnost“. Pro příklad, prostřednictvím třídy *ShowAnchorsOnMouseHover* je spravován obsluhovač události najetí myši na geometrický tvar. Aktivace probíhá pomocí metody *Feature#addHandlers*, deaktivace pomocí *Feature#removeHandlers*. Tyto třídy navíc umožňují nastavení metod zpětného volání (*callback*) pro zpracování dané události externě.

Koncepce zasazení do kontextu aplikace

Komponentami modulu není zajištěna interakce mezi uživatelem a samotnou kreslící tabulí, k těmto účelům je nutné navrhnout a implementovat komponentu vhodnou pro konkrétní aplikaci, což je ponecháno na programátorovi. V případě vyvíjené aplikace jde o tzv. „*Mediator*“, abstraktní třídu deklarovanou v rámci modulu jádra aplikace (viz kapitola 4.4.4). Koncepci lze nicméně popsat alespoň abstraktně pomocí sekvenčního diagramu (viz obr. 4-10).



Obrázek 4-10: Koncepce mediátoru, zdroj: vlastní

4.4.3 Modul doménového modelu

Aby byla zajištěna racionální směrovost závislostí modulů (a nevznikaly nevyřešitelné cirkulární závislosti), byl definován modul doménového modelu (*Domain*). V rámci tohoto modulu jsou definovány komponenty, určené pro přesun dat mezi jednotlivými moduly vyvíjené aplikace. Jinými slovy, v rámci doménového modelu lze definovat průřezové komponenty, které je pak možno využít ve všech modulech.

Model Petriho sítě

Model a logika Petriho sítí je sice implementována již v modulu logiky Petriho sítí, nicméně v tomto případě jde o reprezentaci tříd zmiňovaného modulu z pohledu

frameworku JavaFX. Jedná se tedy o model, který umožňuje například vázání dat a zprostředkovává veškerou komunikaci s implementací Petriho sítě, neobsahuje však žádnou logiku. Tento model lze tedy označit za fasádu implementace Petriho sítě.

Návrh persistentní Petriho sítě

Pro nezávislý převod Petriho sítě do její persistentní formy byl vytvořen abstraktní návrh pomocí rozhraní (např. rozhraní *PersistentNet*, *PersistentPlace* atd.). Persistentní formou Petriho sítě je myšlen například textový dokument ve formátu JSON (viz kapitola 4.4.6), který zajišťuje JSON „*zpracovatel*“ (parser). Petriho síť je pak zpracována pomocí daného zpracovatele, který pro tyto účely využívá svou implementaci abstraktního návrhu persistentní sítě (např. *JsonNet*, *JsonPlace*).

Z hlediska modulu jádra aplikace (viz následující kapitola) je práce s persistentní sítí naprosto nezávislá na její formě, skutečná implementace persistentní sítě je v režii konkrétního modulu převodu Petriho sítí do persistentního formátu (viz kapitola 4.4.6).

4.4.4 Modul jádra aplikace

Jádrem aplikace je myšlen programový modul (*PN Core*), jehož prostřednictvím probíhá integrace modulů vyvíjené aplikace. Moduly lze rozdělit následovně:

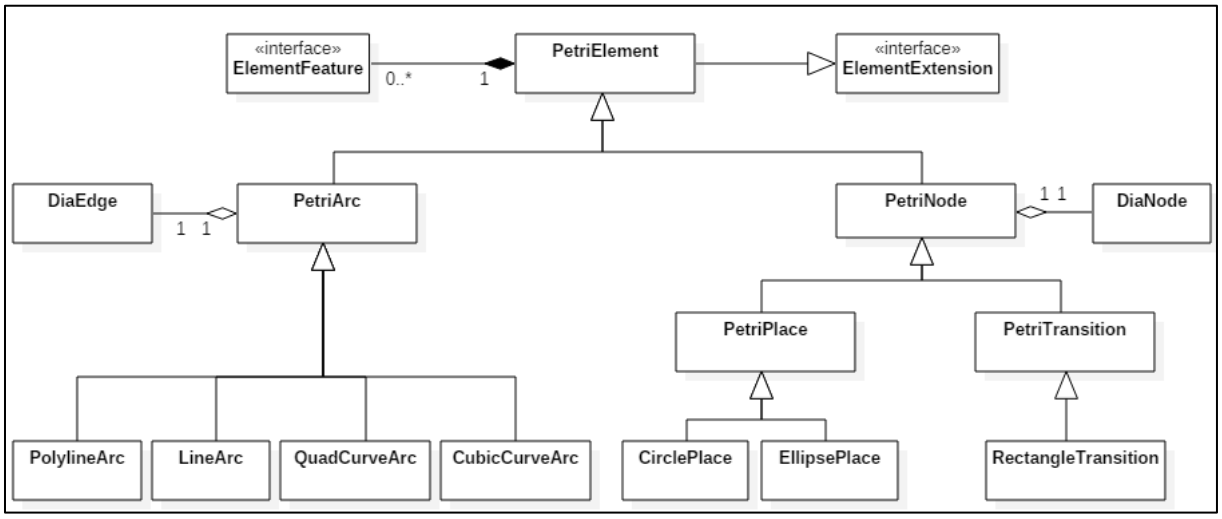
- **moduly závislé na modulu jádra aplikace** – moduly *Desktop Application* a *Mobile Application*,
- **moduly, na kterých je závislý modul jádra aplikace** – moduly *Domain*, *Dao* a jeho implementace, *DataParser* a jeho implementace, *DiaFX*, a *PN Engine* a jeho implementace,
- **programová knihovna frameworku JavaFX**.

UML diagram balíčků je k dispozici na obrázku 4-4 v úvodu kapitoly 4.4.

Modul obsahuje dále výhradně model (logiku) aplikace (z pohledu vzoru MVC), dále však i komponenty, řídící komunikaci (controller) a znovu využitelné komponenty pohledu (view). Z hlediska konkrétního návrhového vzoru jde spíš o hierarchické MVC (HMVC). Pro jednotlivé geometrické tvary a režimy kreslení je totiž definována vlastní triáda MVC. I když nelze hovořit o vzorové aplikaci tohoto vzoru, cíl je však stejný, a to oddělit model (data, logiku) od pohledu v nejvyšším možném rozsahu.

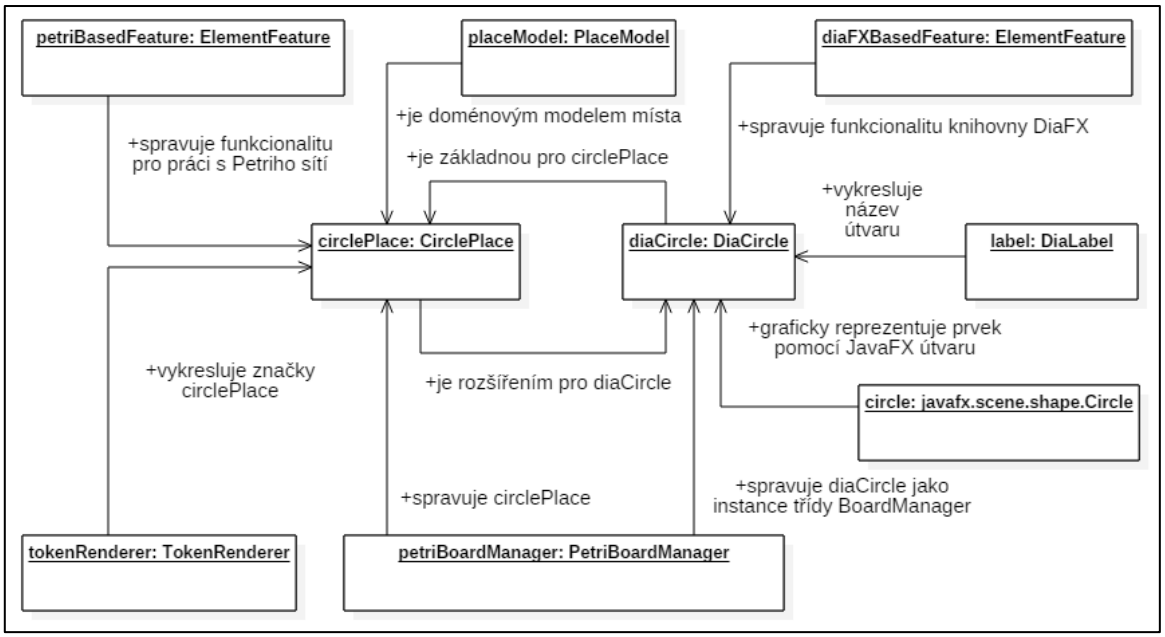
Rozšíření modulu vykreslování diagramů

Modul vykreslování diagramů byl koncipován tak, aby jeho komponenty šlo jednoduše rozšířit pomocí kompozice (viz kapitola 4.4.2). Třídy implementující rozhraní *Extension*, určené pro potřeby vykreslování komponent Petriho sítě (např. vykreslování značení, priorit atd.), se nachází v právě v modulu jádra aplikace a jsou spravovány rozšířením *manažera tabule*, a to *manažerem prvků Petriho sítě* (*PetriBoardManager*). Tyto třídy obsahují zároveň i reference na model komponent Petriho sítě a umožňují tak efektivně odposlouchávat změny vlastností modelu a propagovat je pohledu. Naopak lze pomocí volání metod modelu tyto vlastnosti jednoduše měnit. Diagram tříd paralelní hierarchie třídy *PetriElement* lze nají na obrázku 4-11. „*Základna*“ (*base*) třídy *PetriElement* však na obrázku není uvedena, jde o třídu z hierarchie *DiaElement*. Prostřednictvím základny lze spravovat výsledný geometrický útvar jak pomocí *manažera tabule*, tak i jeho rozšíření *manažera prvků Petriho sítě*.



Obrázek 4-11: Diagram tříd rozšíření modulu diafx - petrifx, zdroj: vlastní

Za účelem doplnění výše uvedené koncepce je na obrázku 4-12 uveden diagram objektů výsledného prvku uživatelského rozhraní – geometrického útvaru *DiaCircle*, rozšířeného pomocí *PetriPlace* a modelu *PlaceModel*.



Obrázek 4-12: Diagram objektů výsledného UI prvku, zdroj: vlastní

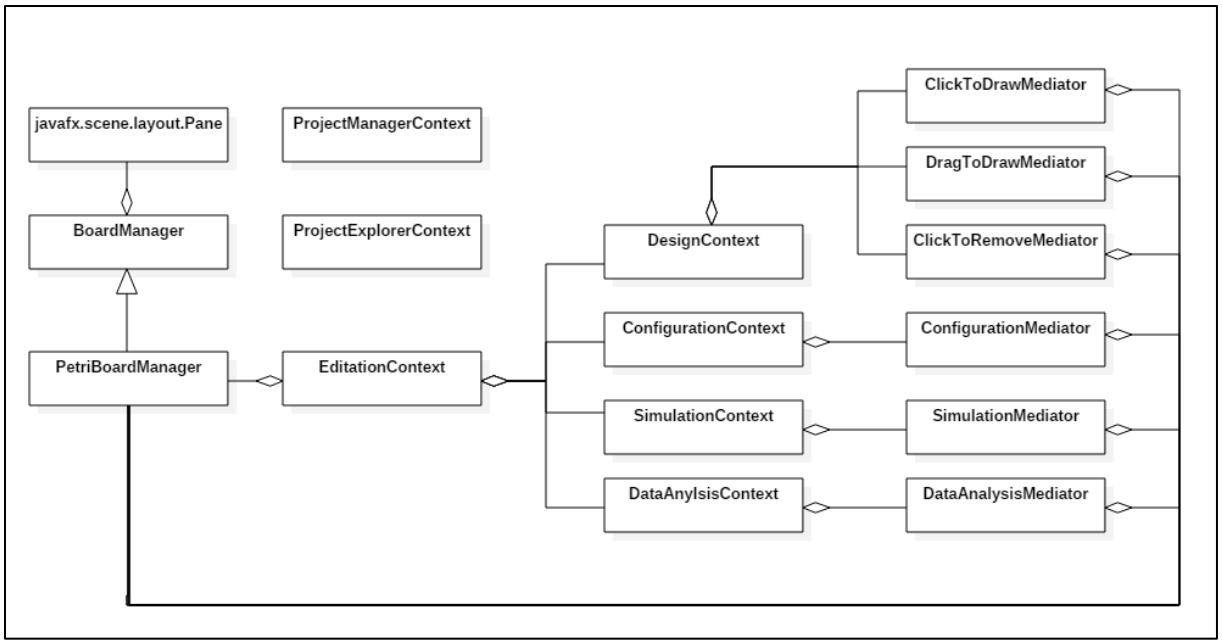
Kontextuální komponenty

Z důvodu, že nelze jednoznačně oddělit znovupoužitelné komponenty tohoto modulu na *model*, *view* a *controller*, byly zavedeny tzv. kontexty – komponenty, které integrují všechny tři výše zmíněné typy komponent. Pro více informací viz diagram tříd na obrázku 4-13.

Složkou typu controller tohoto modulu je spravován hlavně stav celé aplikace, informace spojené s fázemi práce s vyvíjenou aplikací a jednotlivých komponent (např. mediátorů), které si lze ve výsledku spojit s konkrétní implementací prvku GUI na úrovni pohledu (třídy končící na slovo „*Tools*“). Prostřednictvím těchto tříd lze komunikovat například i s moduly přístupu k datům.

Modelovou složkou těchto komponent jsou vlastnosti (*properties*), jejichž změny lze odposlouchávat nebo vázat jejich hodnotu na samotné prvky uživatelského rozhraní, což bude blíže rozebráno v kapitole 4.4.7.

Z komponent typu *view* jde o *kreslící tabuli* – UI prvek třídy *java.scene.layout.Pane* a *manažera prvků Petriho sítě.*

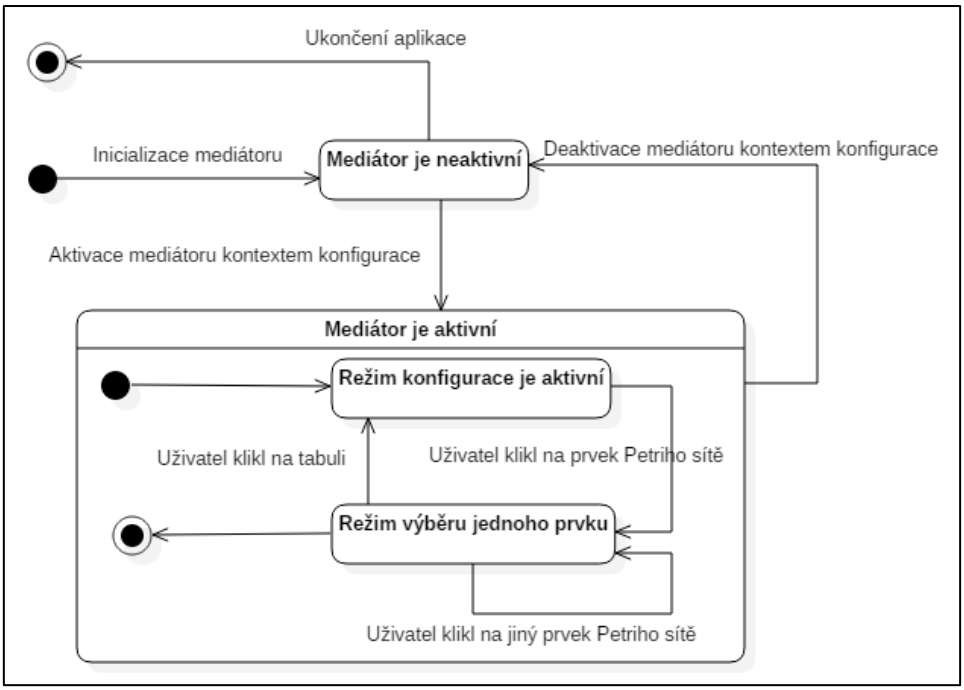


Obrázek 4-13: Diagram tříd kontextu aplikace, zdroj: vlastní

Komponenty typu mediátor

V předchozí kapitole zmiňované třídy typu *Mediator* (*prostředník*) jsou hlavními zástupci řízení komunikace celé vyvíjené aplikace. Pomocí těchto tříd je řízena interakce uživatele a modulu vykreslování diagramů. Pro tyto účely je nestačí propojit jen s instancí třídy *BoardManager*, ale i s modelem GUI (např. panel výběru útvarů) nebo modelem samotné Petriho sítě, a to kvůli správě jejích komponent. Ty jsou reprezentovány jak odpovídající třídou modelu (např. *PlaceModel*), tak i samotným geometrickým útvarem, rozšířením třídy *DiaElement* pomocí kompozice (např. *PetriPlace*).

Stav třídy typu *prostředník* je udržován pomocí jejích datových položek, kterými jsou mimo jiné i instance podtříd abstraktních vnořených tříd, deklarovanou v rámci odpovídajícího rodiče (třídy typu *prostředník*). Na stavovém diagramu (viz obr. 4-14) jsou zachyceny stavy třídy *ConfigurationMediator, režim konfigurace* a *režim výběru jednoho prvku*, které jsou reprezentovány vnořenými třídami *ConfiguringState* a *ElementSelectedState*.



Obrázek 4-14: Diagram stavů mediátoru konfigurace, zdroj: vlastní

Výše popsaný koncept je ve spojení s modely fází a procesů dostatečně škálovatelný a zároveň atomický na takové úrovni, že lze relativně rychle implementovat a přidat novou třídu typu prostředník, a na druhou stranu rozšířit stávající implementaci o zapouzdřenou funkcionalitu pomocí nového stavu.

4.4.5 Moduly přístupu k datům

Důležitou funkcionalitou aplikací je ukládání výsledků práce uživatelů, v případě vyvíjené aplikace jde o správu projektů a jejich Petriho sítí. V rámci API přístupu k datům by tedy měly být deklarovány metody jako „ulož projekt, pozměň projekt, ulož Petriho síť“ a další. Třídy určené pro přístup k datům jsou obvykle nazývány jako DAO (*Data Access Object*) nebo *Repository* (*repositář*). Pro účely této práce bude použito první uvedené jmenné konvence (DAO).

Aby bylo možné případně změnit způsob přístupu k datům – změnit úložiště ze souborového systému disku počítače či tabletu na službu, jako je například DropBox – je vhodné definovat žádanou funkcionalitu API abstraktně pomocí rozhraní. Dále vytvořit pro jednotlivé způsoby ukládání dat samostatné moduly, které budou reprezentovat jejich implementaci. Za předpokladu, že je výše uvedený princip aplikován v praxi, lze komponenty volně vázat na základě vzoru *Dependency Injection* (DI).

Pro účely vyvíjené aplikace byly vytvořeny dva následující moduly:

- **PN Dao** – (*pn-dao*) modul návrhu,
- **Filesystem PN Dao** – (*pn-dao-filesystem*) modul pro práci se souborovým systémem zařízení.

4.4.6 Moduly převodu do persistentních formátů

Moduly přístupu k datům je zajištěna správa dat vyvíjené aplikace, nicméně v jejich rozsahu není řešeno samotné zpracování dat – převody metadat a Petriho sítí do formy, ve které je lze uložit a opětovně načíst.

Pro účely vyvíjené aplikace byly vytvořeny, jako i v předchozím případě, dva následující moduly:

- **PN Parser** – (*pn-parser*) modul návrhu,
- **JSON PN Parser**– (*pn-parser-json*) modul pro převod do formátu JSON.

Alternativou v dnešní době rozšířeného formátu JSON (Javascript Object Notation) jsou například formáty textové reprezentace dat XML a CSV, popřípadě binární formáty jako BSON nebo serializace.

Výhodami ukládání dat ručně v textovém formátu v porovnání se serializací, jsou následující:

- **čitelnost dat jinými aplikacemi** – možnost zpracování dokumentu prostřednictvím balíku aplikací, které jsou založené na rozdílných technologiích,
- **možnost úpravy dat uživatelem**.

Z nevýhod jde například o následující:

- **objem dat** – binární reprezentace dat může uspořit místo tím, že lze opomenout metadata v podobě názvů datových položek a místo toho ukládat data v sekvenci na základě datových typů.

4.4.7 Moduly uživatelského rozhraní

Všechny výše uvedené moduly jsou využitelné jak v desktopové, tak mobilní verzi aplikace. Jedinými moduly, které bylo nutné vytvořit pro danou verzi aplikace,

jsou moduly uživatelského rozhraní. Oficiální cestou autorů projektu Gluon Mobile je tvorba jednoho GUI, které se poté jen přizpůsobí. Pro účely vyvíjené aplikace však bylo rozhodnuto, že moduly budou zvlášť, GUI se značně liší pro odpovídající platformy.

Moduly uživatelského rozhraní jsou zároveň spustitelnými moduly, a proto se nazývají „*Desktop Application*“ a „*Mobile Application*“.

Komponenty modulu

Moduly uživatelského rozhraní obsahují výhradně pohledy (*view*), zpracované ve formátu FXML (viz kapitola 3.2.2) a jejich řídící komponenty (*controller*), kterými je zajišťováno provázání těchto pohledů a modelů z modulu jádra aplikace.

Controller

Komponenty typu controller jsou reprezentovány třídami implementujícími rozhraní *javafx.fxml.Initializable*. Jejich datové položky jsou nastaveny frameworkem JavaFX na základě anotace @FXML a jejich identifikátoru. Jedná se hlavně o prvky uživatelského rozhraní, které jsou definovány v rámci FXML souborů, jejich inicializace je řízena frameworkem. Metoda *Initializable#initialize()* je zavolána po instanciaci třídy, inicializaci prvků uživatelského rozhraní a jejich následném nastavení odpovídajícím datovým položkám.

View

Samotné uživatelské rozhraní je definováno ve formátu FXML, což je rozšíření jazyku XML. Na rozdíl od webových stránek je nabízeno větší množství prvků uživatelského rozhraní (viz kapitola 3.2.2), jejichž vzhled jde pozměnit pomocí kaskádových stylů.

4.5 Omezení plynoucí z možnosti překladu aplikace

Vývoj aplikace nebyl ovlivněn výraznějšími omezeními plynoucími z možnosti překladu aplikace prostřednictvím projektu Gluon Mobile. Na druhou stranu je nutné zmínit alespoň některá úskalí vývoje. Jedná o problémy s výkonem GUI a omezení využití funckionality programovacího jazyka Java 8.

4.5.1 Slabý výkon prvků UI

Překladem pro mobilní platformy vzniká aplikace, která však po stránce grafického rozhraní nemá charakteristiky nativní aplikace. Výsledná mobilní aplikace nevyužívá prvky UI dané platformy. GUI je vykreslováno podobným způsobem jako v případě aplikace využívající čistě grafické akcelerace OpenGL, tedy 2D nebo 3D her. Aplikace tedy občas působí neresponsivně na zařízeních se slabším HW, jde hlavně o animace efektů tlačítek a prvků UI.

Výše uvedený problém se však netýká výkonu kreslící tabule, což je pro vyvinutou aplikaci klíčové.

4.5.2 Java Stream API

Bohužel nelze využívat Java Stream API, které bylo začleněno do programovacího jazyka Java ve verzi 8. V praxi je tedy nutné aplikovat klasický imperativní způsob iterace kolekcemi.

Naopak lambda funkce jsou podporovány. Tyto funkce jsou přeloženy do podoby anonymních tříd prostřednictvím pluginu *RetroLambda*.

4.5.3 Volání neexistujících metod

V průběhu překladu aplikace není validována kompatibilita se staršími verzemi programovacího jazyka Java. Po spuštění nekompatibilní aplikace na mobilním zařízením s OS Android může dojít k pádu aplikace z důvodu runtime výjimky „*NoSuchMethodException*“.

4.5.4 Problém s funkcí InvokeDynamic

Funkce *InvokeDynamic* není podporována, což znemožnilo využít v projektu nejnovější verzi knihovny Google Guava a bylo nutné využít starší verzi. Na druhou stranu je na tento problém upozorněno již při překladu aplikace, který následně končí neúspěchem. Je tedy alespoň zajištěno, že po spuštění aplikace nedojde vyhození runtime výjimky.

4.5.5 Zkušenosti s překladem vyvíjené aplikace pro platformu iOS

Zatímco mobilní verze aplikace pro platformu Android byla s výše uvedenými omezeními a problémy s výkonem úspěšně zkompilována a zprovozněna na tabletu Lenovo Tab3 8, v případě verze aplikace určené pro iPad s iOS nastaly následující problémy:

- v dokumentaci projektu Gluon nebyl nalezen jednoznačný způsob uvedení tzv. „*Provisioning profile*“, aplikaci tedy nebylo možné digitálně podepsat přímo z prostředí NetBeans,
- při následném pokusu o kompilaci digitálně nepodepsané aplikace pomocí daemon služby nástroje Gradle proces trval více než 15 minut a poté skončil vyhozením výjimky *java.lang.OutOfMemoryError*,
- po vyřešení problému s nedostatkem paměti pomocí deaktivace daemon služby nástroje Gradle a navýšením maximální velikosti haldy prostředí NetBeans na 4 GB, proces překladu trval nejméně 20 minut (na zařízení iMac, Intel Core i7), a v jeho průběhu se objevila řada varování na tzv. „*Phantom Class*“ (dále jen fantom třída),
- fantom třídy jsou dle dokumentace původci runtime výjimek *java.lang.ClassNotFoundException*,
- z výše uvedených důvodů byl proces kompilace ukončen.

Východiska překladu aplikace pro iOS

Aplikaci se tedy v tuto chvíli zkompilovat a zprovoznit na zařízení s operačním systémem iOS, i přes propagaci nástroje JavaFXPorts autory Projektu Gluon Mobile, nepodařilo. Jedním z možných důvodů je složitost aplikace, skládá se totiž z více než 350 kompilačních jednotek a obsahuje rovněž více než pět externích knihoven.

Na základě analýzy problému byla stanovena následující možná řešení:

- aplikaci zjednodušit, popřípadě vyčlenit problémové knihovny,
- aplikaci zprovoznit v prostředí Gluon VM (virtuální stroj Gluon), který je dle autorů projektu Gluon Mobile (Gluon, c2017b) ve vývoji.

4.6 Představení aplikace

Vyvíjená aplikace je založena na konceptu workflow tvorby Petriho sítí (viz kapitola 4.4.3), od něhož se také odvíjí struktura této kapitoly. Postupně bude představeno zpracování jednotlivých fází, které mají podobu aplikačních obrazovek. Ve stručnosti bude uvedena i funkcionalita spojená s danou fází workflow, která je validována v kapitole 5.

V textu je myšleno grafickým uživatelským rozhraním (dále jen GUI) buď okno aplikace, v případě desktopové verze nebo „*aktivita*“, která představuje GUI „*jedné*

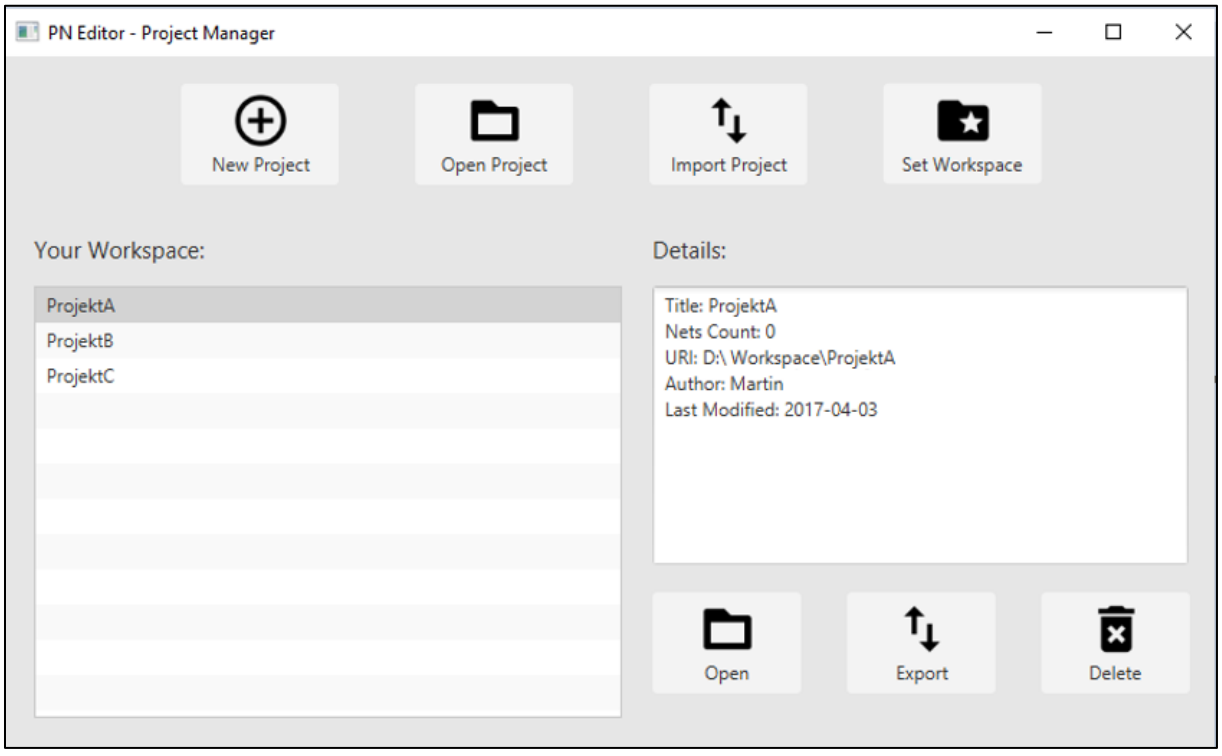
obrazovky“ na mobilních zařízení (OS Android). Jednotlivým fázím byly věnovány následující tři kapitoly.

4.7 GUI fáze výběru projektu

Vstupním bodem aplikace je GUI výběru projektu. GUI umožňuje následující:

- volba umístění pracovního prostoru (workspace),
- zobrazení projektů ve zvoleném pracovním prostoru (workspace),
- založení, otevření, smazání, projektu,
- import/export projektů,
- zobrazení detailů projektů.

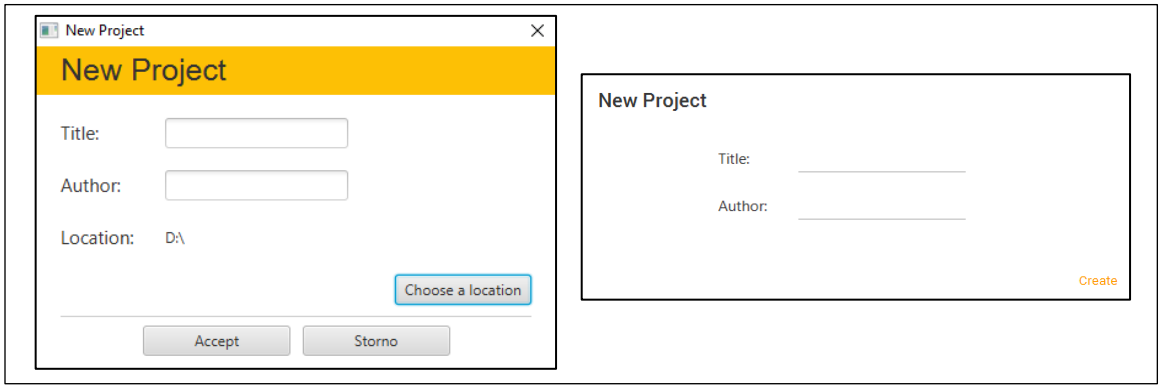
Na obrázku 4-15 je snímek GUI pro desktopovou verzi, mobilní rozhraní je totožné, není tedy uvedeno.



Obrázek 4-15: GUI pro výběr projektů, zdroj: vlastní

4.7.1 Dialogy

Uživatelské rozhraní zahrnuje dialog pro založení nového projektu (viz obr. 4-16). Pro založení projektu je nutné zadat jeho název, autora a cestu k jeho složce, v případě mobilní verze jsou projekty umístěny ve výchozím pracovním prostoru.



Obrázek 4-16: Dialog pro založení projektu, vlevo desktopová verze, vpravo mobilní, zdroj: vlastní

4.7.2 Workspace dle platformy

Práce se souborovým systémem a best-practice uživatelů je závislé na platformě. Zatímco v případě všech desktopových platforem není problém uživatele vyzvat k výběru umístění pracovního prostoru zobrazením okna s výběrem složky, na mobilních zařízeních jsou best-practice odlišné.

Uživatelé OS Android mají možnost přistupovat k souborovým systémům svých zařízení pomocí aplikací vestavěných již v samotném operačním systému. Není problém si pořídit profesionální prohlížeč souborů a mít plnou kontrolu nad úložištěm zařízení. Na druhou stranu zařízení s operačním systémem Apple iOS plně abstrahují od úložiště, aplikace běžně využívají diskový prostor přidělený operačním systémem.

Mobilní verze aplikace má tedy pevně nastavené umístění pracovního prostoru na složku, která jí byla přidělena operačním systémem, což je také standardní umístění v případě desktopové verze.

4.7.3 Sdílení projektů mezi zařízeními

Sdílení projektů probíhá prostřednictvím importu a exportu. V současné vývojové fázi tento způsob plně postačuje, v budoucnu se však nabízí implementace podpory služeb jako je DropBox, Google Disk nebo OneDrive.

Projekt je exportován do formátu zip, na cílovém zařízení lze tento projekt importovat, čímž je vytvořen ve zvoleném pracovním prostoru. Pokud se v pracovním

prostoru nachází projekt o stejném názvu, obsah importovaného projektu nahradí jeho současný obsah v pracovním prostoru.

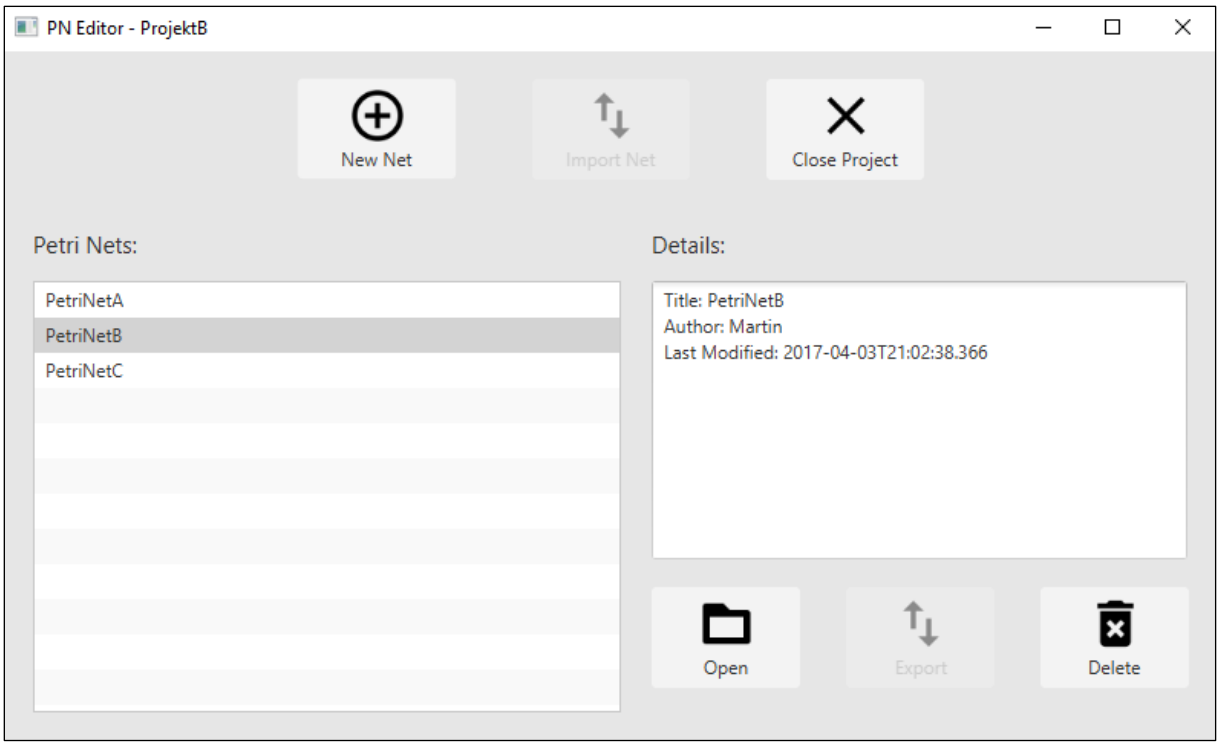
Lokace importovaných a exportovaných projektů je v desktopové verzi řešena výběrem složky, v případě mobilní verze jde o možnost výběru z předpřipravených lokací jako je složka *documents*, *download* nebo *bluetooth*, které se nachází v prvním uložišti označitelném „externí uložiště“, což může být jak interní paměť, tak i vyjímatelné médium jako je SD karta (záleží na typu OS a jeho verzi).

4.8 GUI fáze správy projektu

Po úspěšném výběru projektu, popřípadě založení nového, je uživateli nabídnuto následující:

- zobrazení výčtu sítí vybraného projektu,
- zobrazení detailů vybrané sítě,
- přidání a smazání sítě,
- přesun do režimu práce se zvolenou sítí.

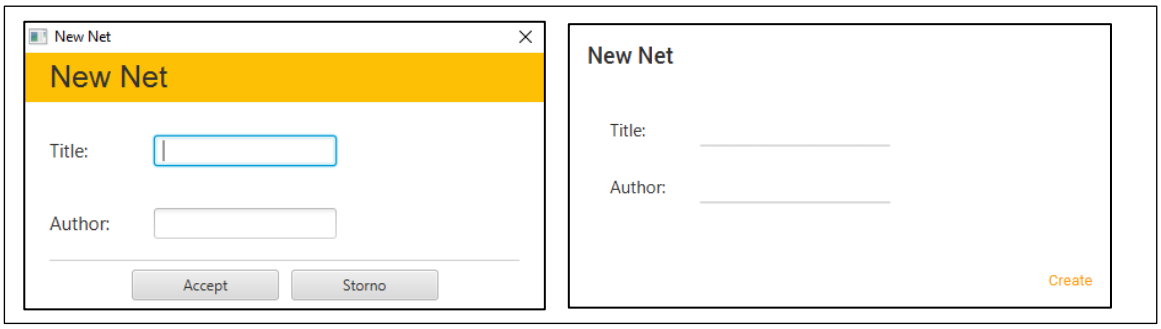
Na obrázku 4-17 je snímek GUI pro desktopovou verzi, mobilní rozhraní je totožně, není tedy uvedeno.



Obrázek 4-17: GUI pro správu projektu, zdroj: vlastní

4.8.1 Dialogy

Uživatelské rozhraní zahrnuje dialog pro vytvoření Petriho sítě (viz obr. 4-18). Pro tvorbu Petriho sítě je nutné uvést její název a autora pro případ, že na jednom projektu pracuje v týmu.



Obrázek 4-18: Dialog pro vytvoření Petriho sítě, zdroj: vlastní

4.8.2 Návrhy pro budoucí vývoj

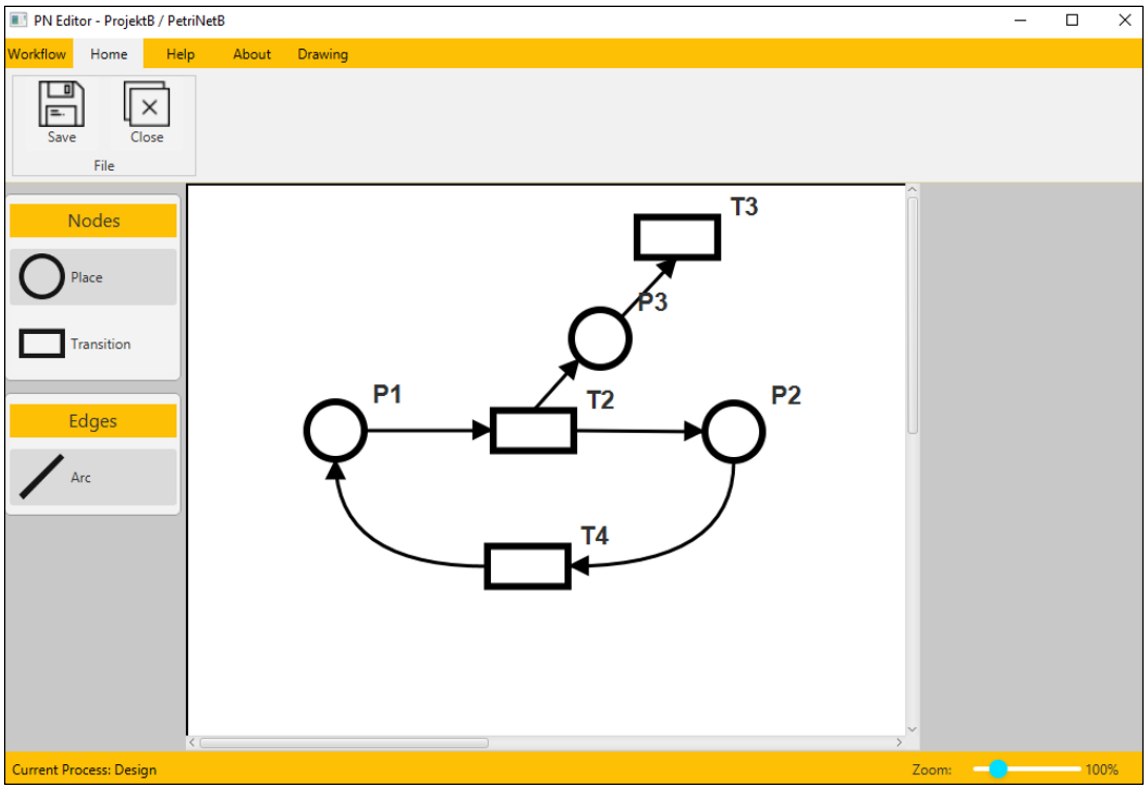
Sítě by mohly být v budoucnu využívány jako komponenty. Takové sítě by pak bylo možné importovat a exportovat a využívat i mimo kontext jejich projektu pomocí reference.

4.9 GUI fáze práce se sítí

Jedná se o hlavní GUI aplikace a bude podrobně rozebráno i z hlediska jeho komponent, sekcí a základní funkcionality, která je validována v kapitole 5. Na obrázku 4-19 je snímek GUI pro desktopovou verzi, na obrázku 4-20 pro mobilní verzi.

GUI desktopové verze zahrnuje následující sekce:

- **pás karet** (*ribbon tab*) – moderní a přehledný prvek uživatelského rozhraní, který je využíván přibližně od roku 2007 a nahrazuje tradiční lištu s nabídkou,
- **levý panel s kontextovou nabídkou aktivního procesu** – každý proces (viz další text) má k sobě přiřazenou kontextovou nabídku, ve které se nachází podpora tohoto procesu ve formě rychle dostupných UI prvků.
- **pravý panel s přehledem vlastností vybrané komponenty sítě** – po výběru komponenty sítě (například přechodu) jsou v tomto panelu zobrazeny její vlastnosti, oddělené pomocí tzv. akordeonu (*accordion*),
- **dolní lišta** – klasická stavová lišta, na níž je uveden aktivní proces, popřípadě další informace o stavu aplikace.



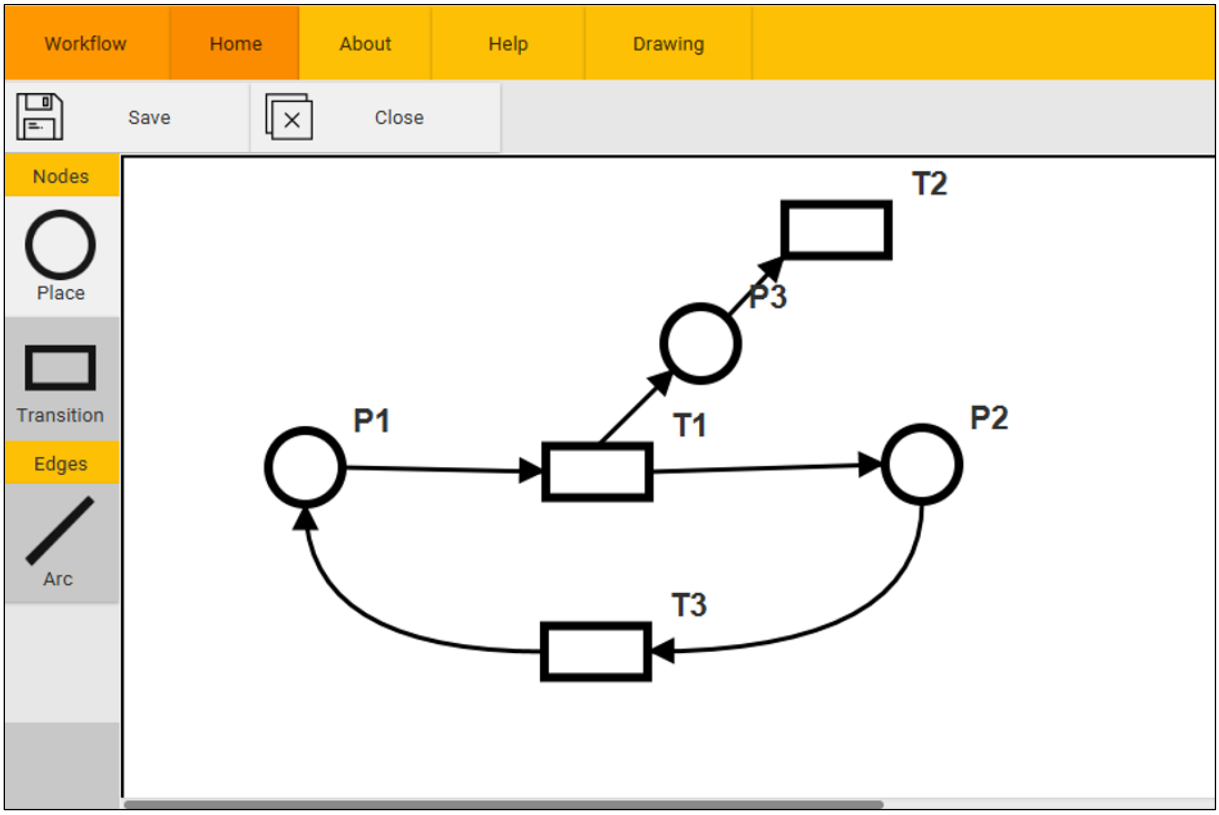
Obrázek 4-19: GUI dektopové verze, zdroj: vlastní

Optimalizované GUI pro mobilní verze zahrnuje následující sekce:

- **pás karet** – minimalistická verze pásu karet GUI desktopové verze,
- **levý panel s kontextovou nabídkou aktivního procesu** – užší varianta panelu.

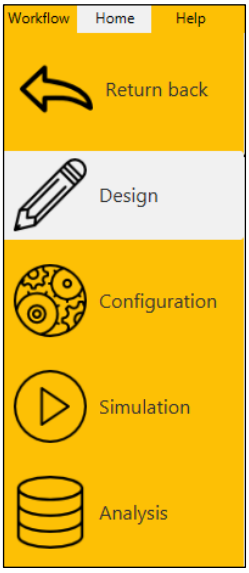
Chybějící sekce byly nahrazeny následovně:

- **pravý panel s přehledem vlastností vybraného komponentu sítě** – řešení pomocí dialogu s možnostmi nastavení vlastností pomocí adekvátních prvků UI,
- **dolní lišta** – stavová lišta není v mobilní verzi k dispozici.



Obrázek 4-20: GUI mobilní verze, zdroj: vlastní

Následuje představení jednotlivých procesů práce se sítí (viz kapitola 4.3.3). Na procesy je v dalším textu odkazováno jako na „režimy“. V rámci GUI lze mezi režimy libovolně přecházet prostřednictvím nabídky (viz obr. 4-21) (dále workflow panelu), která se zobrazí na levé straně obrazovky po kliknutí na tlačítko „Workflow“ v levém horní rohu. Nejde však o vyčerpávající přehled veškeré funkcionality.



Obrázek 4-21 : Workflow Panel, zdroj: vlastní

Možností práce s kreslící tabulí

Aby bylo možné plně využít kreslící tabuli, je k dispozici funkcionalita přiblížení/oddálení (zoom) a posouvání viditelné oblasti kreslící tabule pomocí grafického prvku posuvného panelu (ScrollPane).

Za předpokladu, že uživatel používá pro kreslení myš, je zoom nastavitelný pomocí posuvníku v uživatelského rozhraní vpravo dole. Pokud je k dispozici dotykové zařízení, přiblížení/oddálení lze docílit standartním multidotykovým gestem – roztažení dvou prstů.

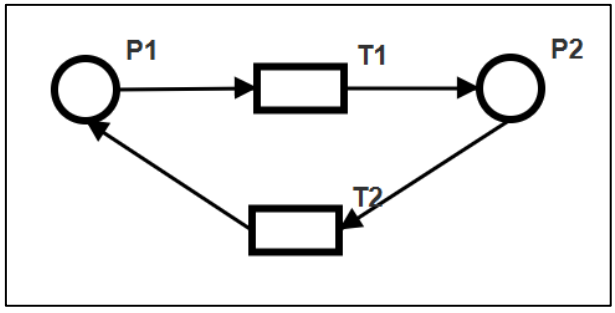
Posun viditelné plochy lze za použití myši provést pomocí posunovacích lišt po krajích kreslící tabule, nebo posunem samotného geometrického útvaru ke kraji viditelné oblasti. Pro posun viditelné oblasti na dotykových zařízeních je nutné provést multidotykové gesto současného posunu dvěma prsty proti směru posunu viditelné oblasti.

Stránkování plochy na kreslení a export do formátu PNG

Kreslící tabule není nekonečně velká, nýbrž odstránkovaná tak, aby bylo možno jednotlivé stránky vyexportovat ve formě obrázku (dále snímek), a ty mohly být vloženy např. do textového dokumentu. V současné době je pro demonstraci funkcionality nastaven formát A4 s tím, že jsou brány v potaz i okraje stránky. Vytištění snímku na stránku A4 tedy není problémem. Snímky jsou generovány ve formátu PNG.

4.9.1 Grafický návrh sítě

Prvním krokem uživatele je obvykle grafický návrh vybrané Petriho sítě. V první fázi tohoto procesu se ještě obvykle nejedná o Petriho síť (ve smyslu definice uvedené v kapitole 2.5.4), nýbrž o obecnou síť (viz kapitola 2.5.3). Vlastnosti míst, přechodů a hran, tj. např. jejich značení, priority nebo násobnosti, nejsou zatím brány v potaz (viz obr. 4-22).

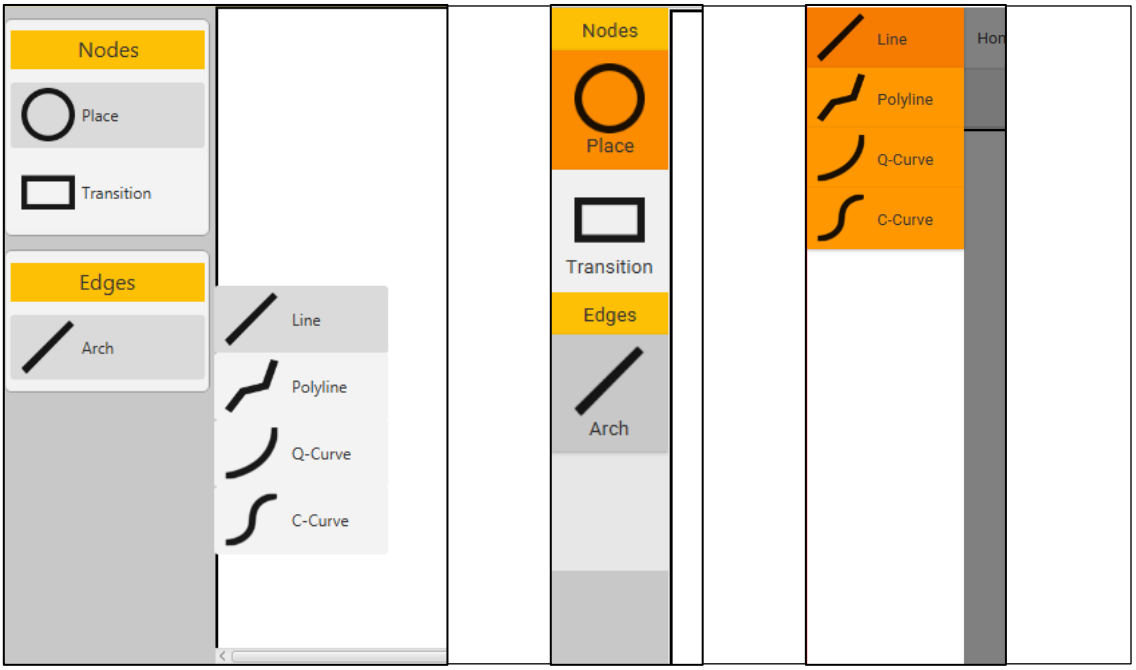


Obrázek 4-22: Síť, zdroj: vlastní

V této fázi návrhu Petriho sítě jde zejména o to, aby byly jednotlivé kroky odděleny a vždy byla k dispozici odpovídající nabídka tak, že ji uživatel nemusí pracně hledat. Motivací oddělení režimu grafického návrhu a konfigurace byla i nutná úspora prostoru a omezení počtu prvků UI pro mobilní zařízení. Tato koncepce umožňuje mimo jiné chytře nabízet to, co uživatel potřebuje.

Výběr a přidání prvku sítě

Geometrické útvary (prvky sítě) jsou rozděleny do dvou kategorií – uzly (*nodes*) a hrany (*edges*). Ty jsou dále rozděleny na prvky typu místa (*place*), přechodu (*transition*) a hrany ve smyslu Petriho sítí (*arc*). Panel nabídky těchto prvků se nachází v uživatelském rozhraní vlevo. Na obrázku 4-23 jsou uvedeny podoby panelu nabídky prvků a také nabídka podoby prvků.



Obrázek 4-23: Panel pro výběr geometrického útvaru (prvku sítě), vlevo: desktopová verze, vpravo: mobilní verze a výsuvný panel, zdroj: vlastní

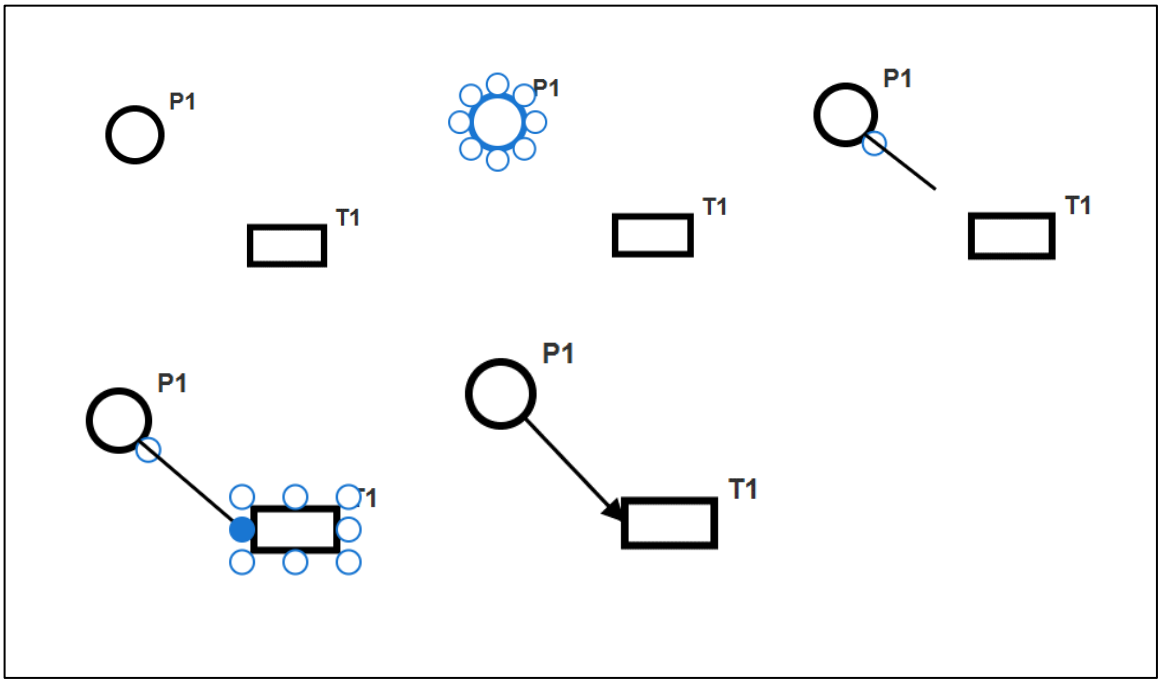
Režimy kreslení

V rámci režimu grafického návrhu sítě jsou k dispozici tři následující režimy kreslení:

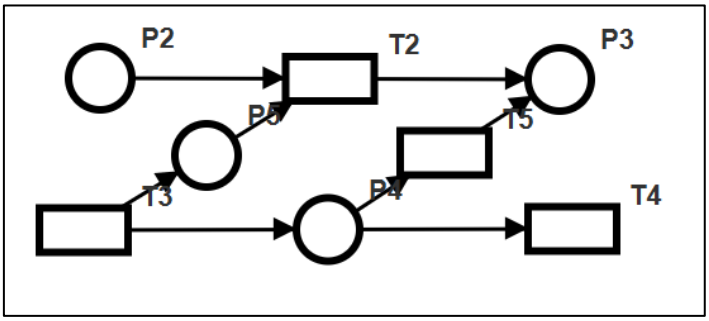
- **kreslení klikem/dotykem** – po kliknutí na kreslící tabuli uživatel nakreslí geometrický útvar. Při najetí myší na daný útvar (v mobilní verzi je nutné

daný útvar označit) lze pomocí tzv. kotevních bodů začít kreslit hranu, kterou lze navázat na kotevní body uzlu patřičné kategorie (viz obr. 4-24),

- **kreslení tahem** – v režimu kreslení tahem lze jedním tahem nakreslit trojici prvků (místo-přechod-místo nebo přechod-místo-přechod), které jsou již spojeny hranami. Tah, který začal na prvku, nebo skončil na prvku propojí hranami daný prvek s dvojicí prvků vzniklých v rámci tahu (viz obr. 4-25), v rámci tohoto režimu lze i upravovat vlastnosti prvků, např. jejich názvy,
- **odstranění prvku klikem/dotykem** – při kliknutí nebo doteku na daný uzel nebo hranu je daný prvek odstraněn. Při odstranění uzlu dochází k odstranění všech hran začínajících nebo končících v tomto uzlu – hrana vždy musí začínat a končit v některém z uzlů.



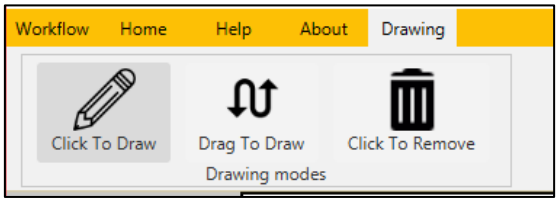
Obrázek 4-24: Režim klik/dotyk pro kreslení prvku, zdroj: vlastní



Obrázek 4-25: Síť nakreslená v režimu kreslení tahem pomocí 4 tahů, zdroj: vlastní

Volba režimu

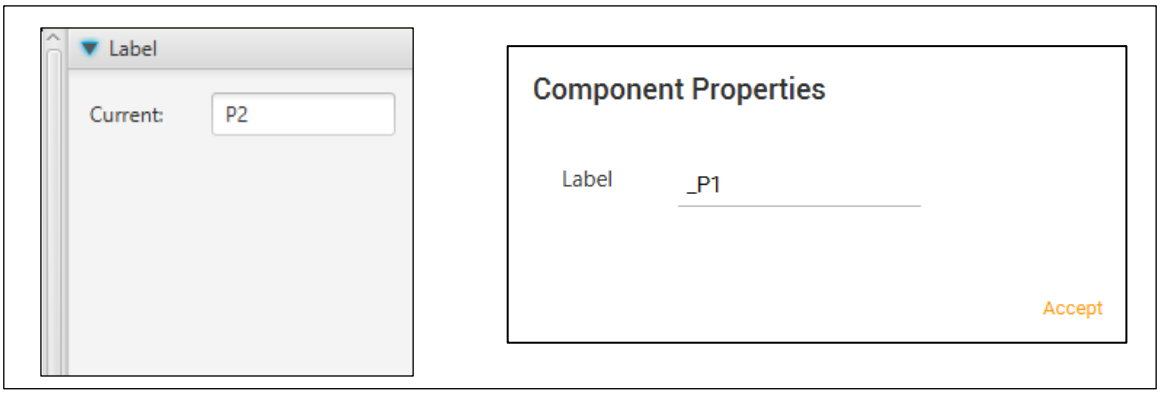
Volba režimu probíhá pomocí záložky kreslení (drawing), která je uvedena na obrázku 4-26. Z tohoto panelu lze vybrat jeden z režimů (viz předchozí text).



Obrázek 4-26: Záložka kreslení, zdroj: vlastní

Vlastnosti prvků

V seznamu režimů kreslení byl uveden název prvku jako jeho vlastnost. Změna názvu probíhá pomocí panelu nabídky vybraného prvku v uživatelském rozhraní vpravo, v případě mobilní verze jde o dialog, který se stejně jako panel zobrazuje po označení daného prvku (viz obr. 4-27).



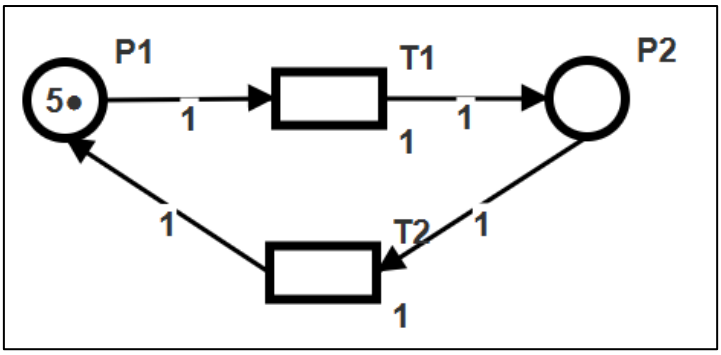
Obrázek 4-27: Nabídka pro změnu vlastností prvků, vlevo desktopová verze - panel, vpravo mobilní verze - dialog, zdroj: vlastní

4.9.2 Konfigurace a validace PN

V okamžiku, kdy uživatel navrhl síť, přesune se pomocí workflow (viz obrázek 4-21) panelu do režimu konfigurace. V tomto režimu lze upravovat vlastnosti komponent dle definice procesních P/T Petriho sítí:

- **místa** – typ (obyčejné, místo zdrojů, vstupní, výstupní), značení,
- **přechody** – priorita,
- **hrana** – násobnost hrany.

V tuto chvíli jsou již zobrazeny všechny výše popsané vlastnosti přímo na tabuli, v blízkosti odpovídajícího prvku (viz obrázek 4-28).

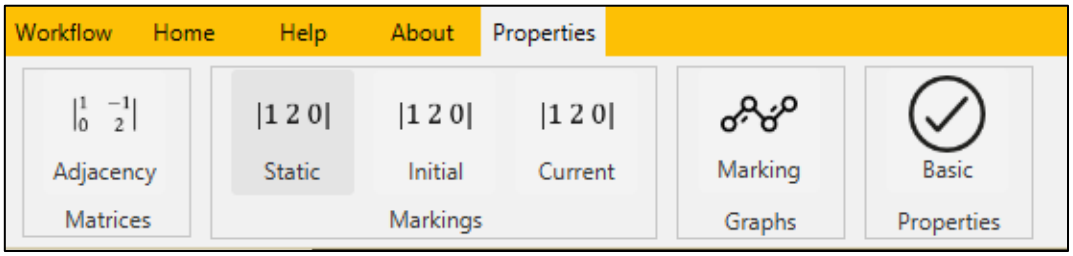


Obrázek 4-28: Konfigurace PPTN, zdroj: vlastní

V rámci tohoto režimu je k dispozici generátor sekvenčního grafu dosažitelných značení, jehož prostřednictvím jsou zjišťovány vybrané vlastnosti Petriho sítě (viz kapitola 2.7.1). Dále je možné zobrazit incidenční matici, počáteční, statické a aktuální značení. Všechny uvedené funkcionality jsou validovány v páté kapitole.

Záložka vlastností PN

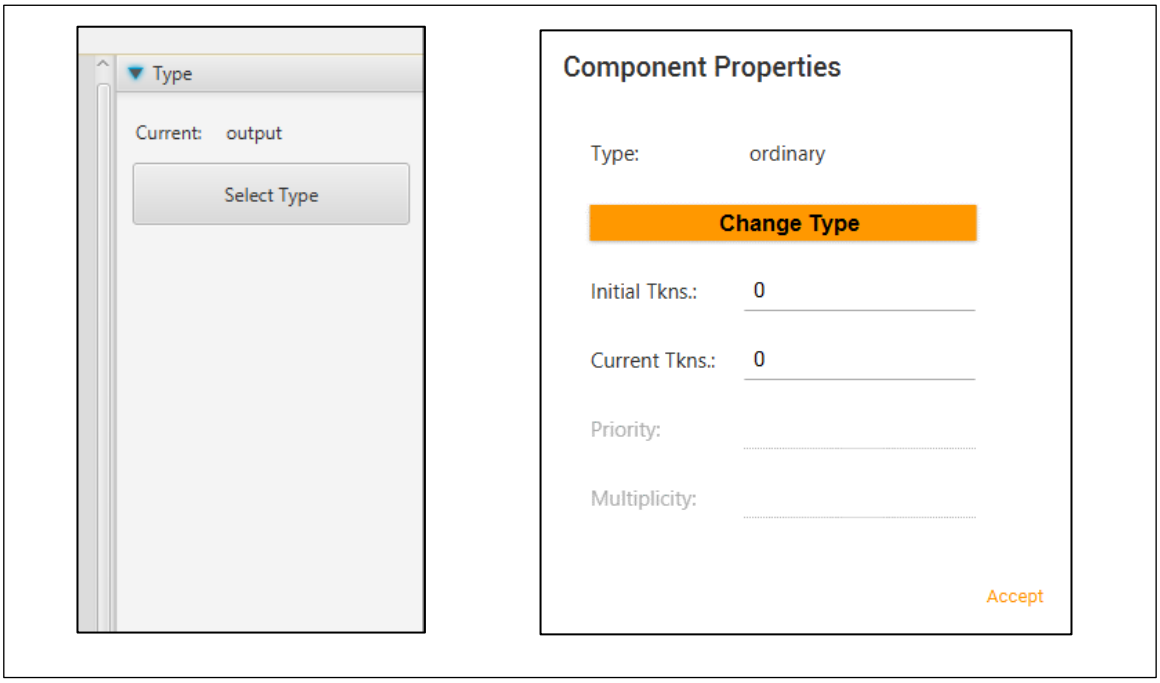
V záložce vlastností (viz obr. 4-29) se nachází všechny UI prvky, které jsou potřeba pro funkcionalitu uvedenou v předchozím odstavci.



Obrázek 4-29: Záložka vlastností, zdroj: vlastní

Panel s vlastnostmi prvku PN

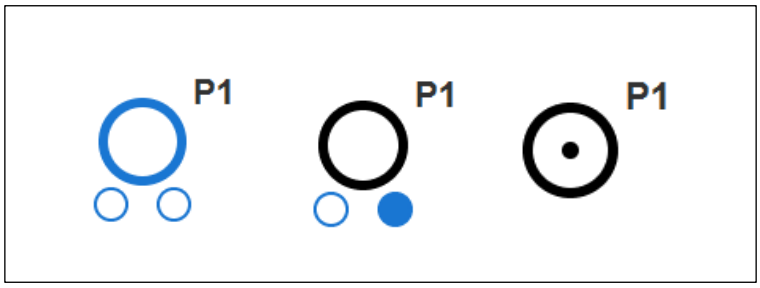
Vlastnosti, uvedené v úvodu této kapitoly, lze nastavovat po kliknutí na daný prvek, prostřednictvím panelu, popřípadě dialogu vlastností prvku Petriho sítě. Pro příklad je na obrázku uveden panel s nabídkou vlastností přechodu a dialog se stejným obsahem pro mobilní verzi (viz obr. 4-30).



Obrázek 4-30: Vlastnosti komponenty PPTN, zdroj: vlastní

Nastavení počtu značek

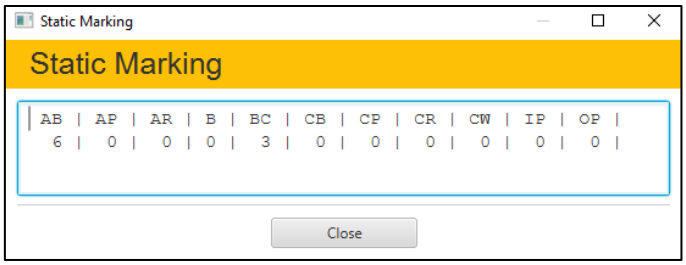
Nastavení počtu značek vybraného místa lze provést prostřednictvím panelu nabídky vlastností nebo pomocí modrých bodů pod dolním okrajem geometrického útvaru (viz obr. 4-31). Princip zobrazování a skrývání bodů je stejný jako v případě kotevních bodů (viz kap. 4.9.1).



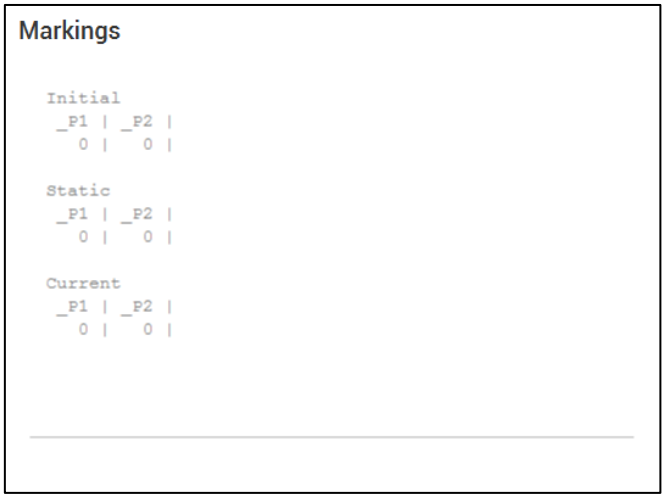
Obrázek 4-31: Nastavení počtu značek, zdroj: vlastní

Textový dialog

Vícenásobně použitelný textový dialog pro zobrazování textové informace (viz obr. 4-32 a 4-33) je určený např. k vykreslení incidenční matice nebo značení.



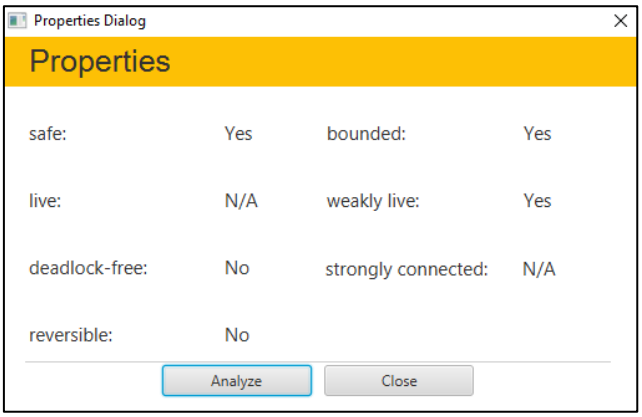
Obrázek 4-32: Textový dialog desktopové verze, zdroj: vlastní



Obrázek 4-33: Textový dialog mobilní verze, zdroj: vlastní

Dialog vlastností

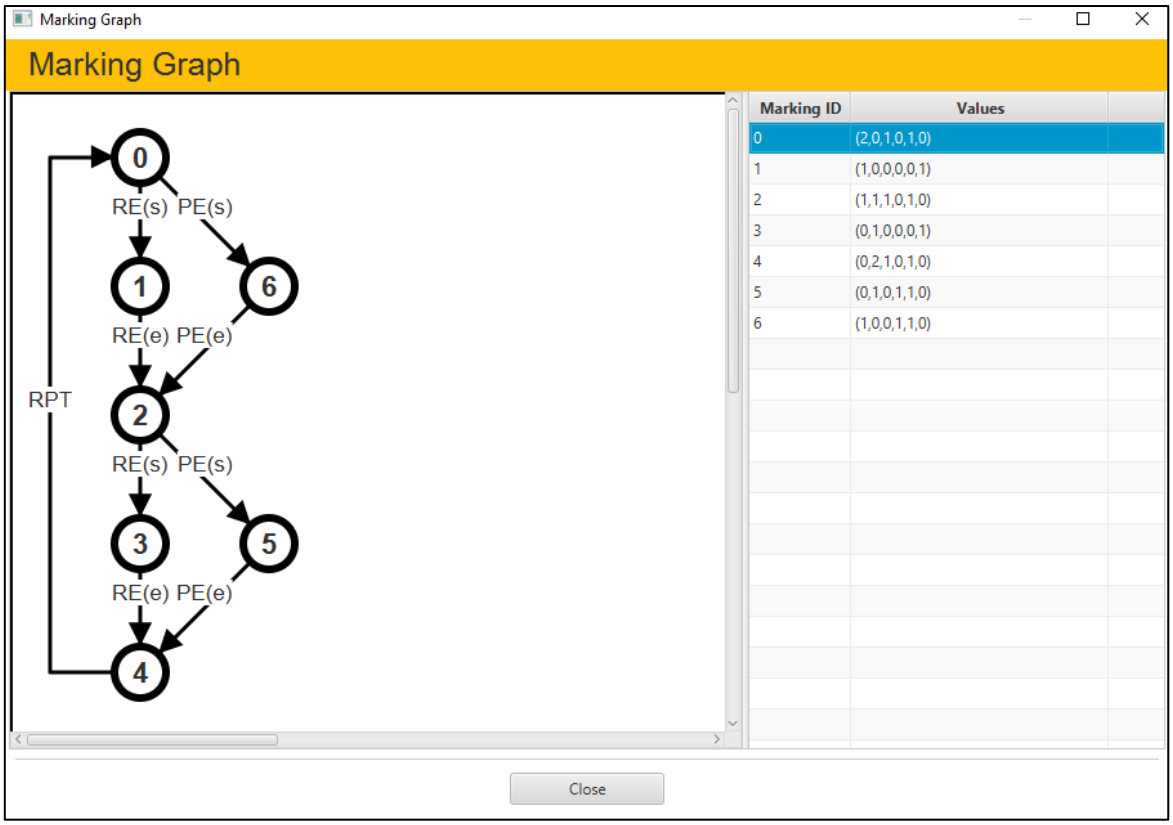
Vlastnosti Petriho sítě lze zkoumat prostřednictvím dialogu vlastností (viz obr. 4-34).



Obrázek 4-34: Dialog základních vlastností, zdroj: vlastní

GUI sekvenčního grafu dosažitelných značení

Desktopové uživatelské rozhraní sekvenčního grafu dosažitelných značení je uvedeno na obrázku 4-35. Mobilní verze tohoto GUI je uvedena na obrázku 4-36.



Obrázek 4-35: Sekvenční graf dosažitelných značení, zdroj: vlastní

Obrázek 4-36: Sekvenční graf dosažitelných značení, zdroj: vlastní

Uzly a hrany jsou reprezentovány stejně jako komponenty Petriho sítí pomocí rozšíření funkcionality modulu DiaFX. Jde o hierarchii tříd *net.cemartinapps.pn.core.graphs.elements.MarkingElement*.

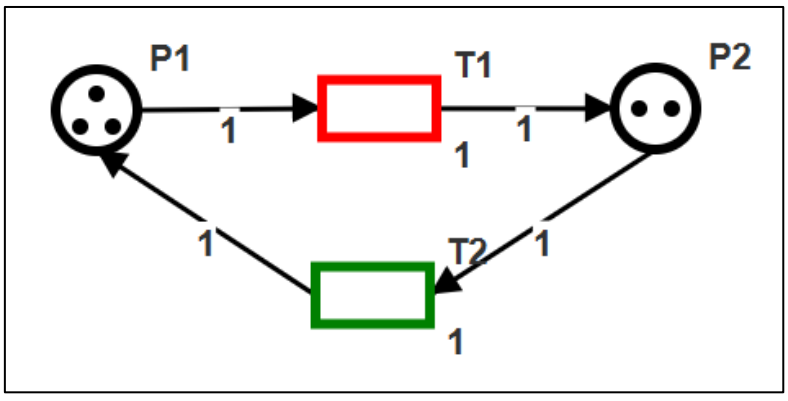
Návrhy pro budoucí vývoj

Pomocí aplikace by mohla být v budoucnu podrobněji zkoumána omezenost sítě. K dispozici by mohl být výpočet p-invariantů nebo vygenerování grafu pokrytí.

4.9.3 Simulace PN

Sekvenční běh navržené a nakonfigurované Petriho sítě lze simulovat v režimu simulace, a to dvěma způsoby:

- **simulace v reálném čase** – sekvenční běh probíhá v reálném čase, počet značek v jednotlivých místech se mění dle jejich aktuální hodnoty a barvy okrajů přechodů (obdélníků) jsou měněny v závislosti na současném stavu přechodu:
  - **normální** – černý okraj,
  - **proveditelný** – zelený okraj,
  - **proveditelný i po zahrnutí prioritní funkce přechodů** – oranžový okraj,
  - **právě provedený** – červený okraj (viz obr. 4-37).
- **provedení běhu na pozadí s uvedením limitu kroků** – sekvenční běh je proveden na pozadí, dále jen „*immediate*“ režim.



Obrázek 4-37: Snímek simulace, zdroj: vlastní

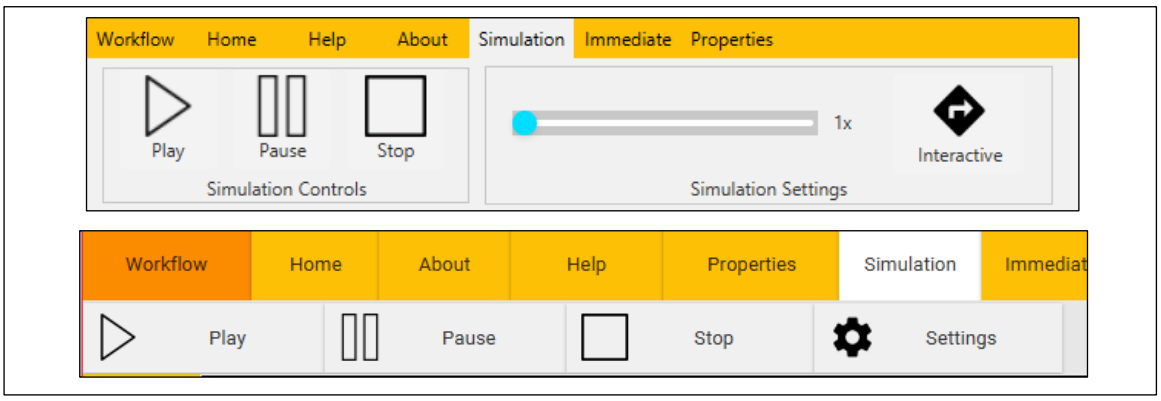
Výsledky obou režimů simulace lze analyzovat v režimu analýzy dat, kterému je věnován následující text.

Záložka simulace v reálném čase

Nabídka režimu simulace v reálném čase zahrnuje následující prvky:

- trojici tlačítek spustit (play), pozastavit (pause) a zastavit (stop),
- posuvník s nastavením rychlosti simulace,
- tlačítko, kterým se volí interaktivní a neinteraktivní režim.

Obsah záložky je uveden na obrázku 4-38. Mobilní verze se liší tím, že rychlost simulace a interaktivní režim se volí prostřednictvím dialogu po kliknutí na tlačítko nastavení (settings).



Obrázek 4-38: Záložka simulace, desktopová a mobilní verze, zdroj: vlastní

4.9.4 Analýza dat

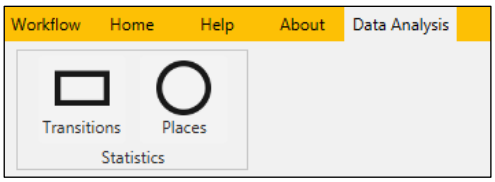
Koncepce režimu analýzy dat spočívá v možnosti analýzy posledního sekvenčního běhu Petriho sítě, v jehož průběhu jsou generovány vybrané statistiky. V současné verzi aplikace jsou k dispozici následující statistiky:

- **základní statistika míst** – průměrný počet značek náležících danému místu a počet kroků, kdy se v místě nenacházela alespoň jedna značka,
- **základní statistika přechodů** – počítadlo proveditelnosti přechodu a počítadlo provedení přechodu.

V budoucnu se očekává přidání nových a žádaných metod analýzy dat s tím, že mohou být i graficky znázorněny přímo v okolí odpovídajícího prvku Petriho sítě.

Záložka statistik

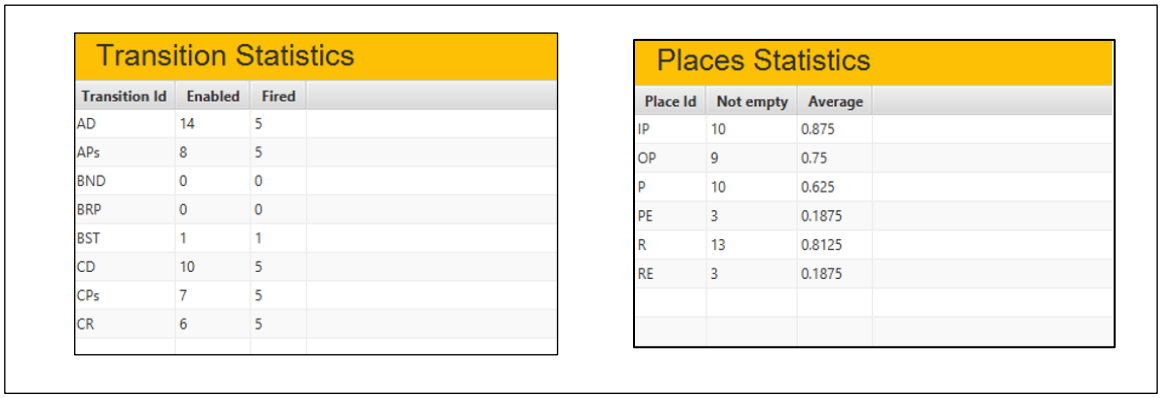
Prostřednictvím záložky statistik, je možné zobrazit dialogy statistik přechodů a míst (viz obrázek 4-39).



Obrázek 4-39: Záložka statistik, zdroj: vlastní

Dialog statistiky míst a přechodů

Dialog statistik (viz obrázek 4-40) je tvořen tabulkou se sloupci odpovídajícími požadovaným statistikám. Řádky reprezentují buď samotné přechody, nebo místa.



Obrázek 4-40: Dialogy statistik, zdroj: vlastní

5 Validace vyvíjené aplikace v praxi

Účelem předchozích kapitol byl rozbor vyvíjené aplikace z pohledu její architektury a finální podoby. Předmětem této kapitoly je však její validace, v jejímž rámci bylo ověřeno, jestli aplikace umožňuje zpracovat jednoduchý ukázkový proces pomocí procesní P/T Petriho sítě (dále jen PPTN) tak, že je uživatel PPTN může graficky navrhnout, konfigurovat, simulovat a v neposlední řadě zjistit její základní vlastnosti.

V následujících podkapitolách byla aplikace validována na procesu práce se dvěma sdílenými systémovými prostředky a jednoduché výrobní lince.

Orientované hrany Petriho sítí jsou v rámci této kapitoly značeny následovně – hrana s počátkem v komponentě (místu nebo přechodu) *A* a ukončením v komponentě *B* je značena (*A*, *B*).

5.1 Proces práce se dvěma sdílenými systémovými prostředky

Sdílení systémových prostředků patří mezi základní problémy v oblasti IT. V případě, že jde jen o jeden systémový prostředek, ke kterému je požadován ze strany programového vlákna nebo procesu (dále jen proces) tzv. výhradní přístup, je procesu přidělen zámek tohoto systémového prostředku a poté může být tento systémový prostředek v rámci daného procesu využit. Pokud je stejný systémový prostředek požadován i jiným procesem, je tento proces blokován, resp. tento prostředek není možné využít.

V situaci, kdy je procesem požadován výhradní přístup k více než jednomu systémovému prostředku, může dojít k situaci tzv. deadlocku (uváznutí, vzájemné čekání) s tím, že v tomto případě jsou procesy vzájemně blokovány.

Stav vzájemného čekání lze řešit např. časovým intervalem (timeout), ve kterém je pravidelně ověřeno, jestli se procesy do tohoto stavu nedostaly. Tento princip je v praxi využíván například v databázových systémech za předpokladu, že podporují transakce, z jejichž strany může docházet k zamykání dat. Mnohdy je však stav deadlock pro daný programový systém fatální.

Vyvíjená aplikace byla validována právě na procesu, v jehož rámci dochází k práci se dvěma sdílenými zdroji. Jde o úpravu záznamu akademického článku (academic paper), dále jen článek, a jeho recenze (review). Článek nesmí být upraven,

v případě, že započala jeho recenze (záznam o recenzi vniká již v průběhu tvorby článku). Za předpokladu, že se změní stav recenze, změní se i stav článku. Všechny akce musí proběhnout atomicky, jako například finanční transakce, kde by byly podobně zamykány dva bankovní účty. Aplikací v praxi, kdy je nutné zamknout více prostředků, je samozřejmě velké množství.

Účelem této kapitoly je validovat, zda za pomocí vyvinuté aplikace lze modelovat procesní P/T Petriho síť programového systému, na jejímž základě pak lze odhalit deadlock v rámci tohoto systému a pokusit se tento stav vyřešit změnou návrhu programového systému (resp. jeho PPTN).

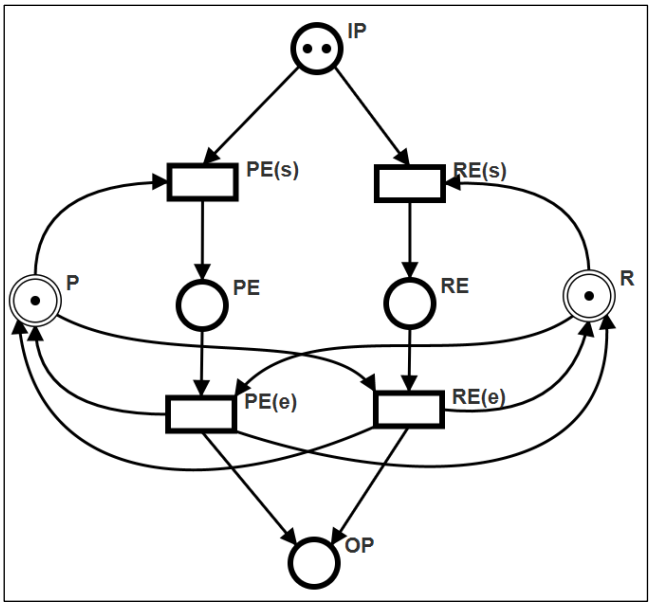
5.1.1 PPTN programového systému

Výčet komponent PPTN je uveden v tabulce 5-1.

Tabulka 5-1: Komponenty PPTN, zdroj: vlastní

Zkratka	Kategorie	Název	Popis
IP	Vstupní m.	Input Place	Počátek procesu
PE(s)	Přechod	Editation of Paper Started	Editace článku začala
RE(s)	Přechod	Editation of Review Started	Editace recenze začala
P	M. zdrojů	Paper	DB záznam článku
PE	Místo	Paper is being edited	Článek je editován
RE	Místo	Review is being edited	Recenze je editována
R	M. zdrojů	Review	DB záznam recenze
PE(e)	Přechod	Editation of Paper Ended	Editace článku skončila
RE(e)	Přechod	Editation of Review Ended	Editace recenze skončila
OP	Výstupní m.	Output Place	Konec Procesu

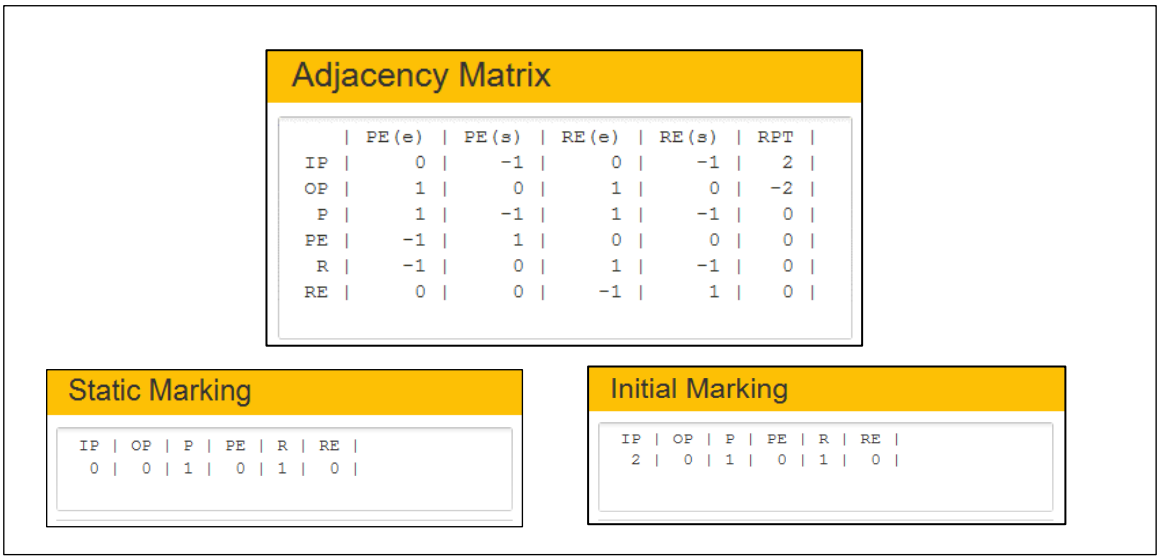
Samotná PPTN je uvedena na obrázku 5-1.



Obrázek 5-1: PPTN modelovaného systému, zdroj: vlastní

5.1.2 Incidenční matice a značení PPTN

Na obrázku 5-2 je uvedena incidenční matice PPTN, statické a vstupní (počáteční značení).



Adjacency Matrix

	PE(e)	PE(s)	RE(e)	RE(s)	RPT
IP	0	-1	0	-1	2
OP	1	0	1	0	-2
P	1	-1	1	-1	0
PE	-1	1	0	0	0
R	-1	0	1	-1	0
RE	0	0	-1	1	0

Static Marking

IP	OP	P	PE	R	RE
0	0	1	0	1	0

Initial Marking

IP	OP	P	PE	R	RE
2	0	1	0	1	0

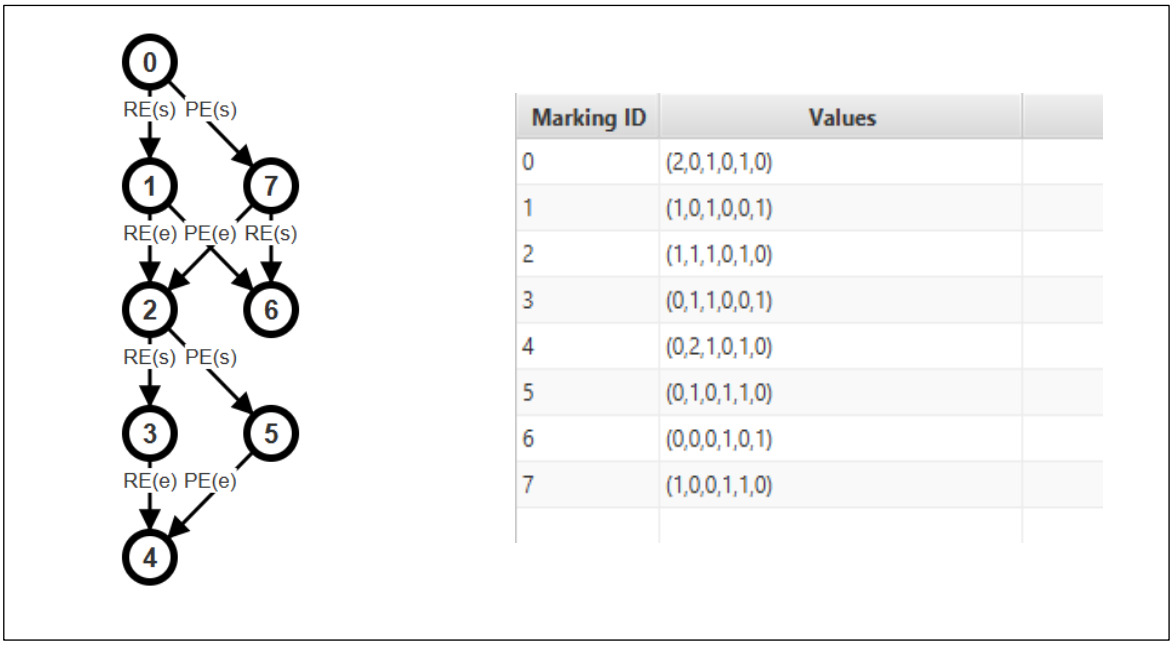
Obrázek 5-2: Incidenční matice a značení PPTN, zdroj: vlastní

5.1.3 Graf dosažitelných značení

Z obrázku 5-3 jasně vyplývá, že ve značení M_6 se síť nachází ve stavu uváznutí, protože se nejedná o výstupní značení (viz obr. 5-2), kdy značky náleží jen výstupnímu místu a místům zdrojů. Naopak jde o stav, kdy nelze přejít do dalšího značení a značky se nachází jen v obyčejných místech (viz následující kapitola).

$$\mathrm{M}_x = (|IP|, |OP|, |P|, |PE|, |R|, |RE|)$$ *Vzorec 5-1*

$$\mathrm{M}_0 = (2, 0, 1, 0, 1, 0)$$ *Vzorec 5-2*

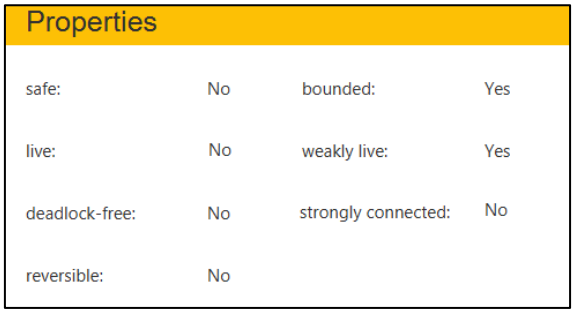


Marking ID	Values
0	(2,0,1,0,1,0)
1	(1,0,1,0,0,1)
2	(1,1,1,0,1,0)
3	(0,1,1,0,0,1)
4	(0,2,1,0,1,0)
5	(0,1,0,1,1,0)
6	(0,0,0,1,0,1)
7	(1,0,0,1,1,0)

Obrázek 5-3: Sekvenční graf dosažitelných značení, zdroj: vlastní, pořadí míst v rámci značení je určeno vzorcem 5-1, pro příklad viz vzorec 5-2

5.1.4 Základní vlastnosti

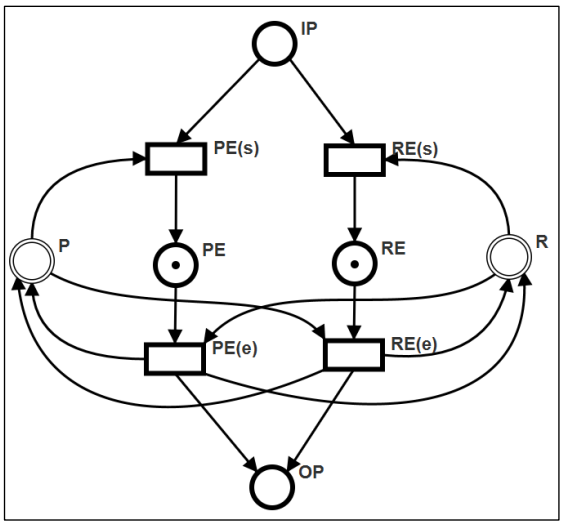
Na základě obrázku 5-4 (dialogu základních vlastností) je daná PPTN pouze omezená a slabě živá. V minulých kapitolách bylo ověřeno, že síť obsahuje deadlock, což lze usoudit i na základě zmiňovaného dialogu základních vlastností – vlastnost „*deadlock-free*“ nabývá hodnoty „*no*“.



Obrázek 5-4: Dialog základních vlastností, zdroj: vlastní

5.1.5 Simulace

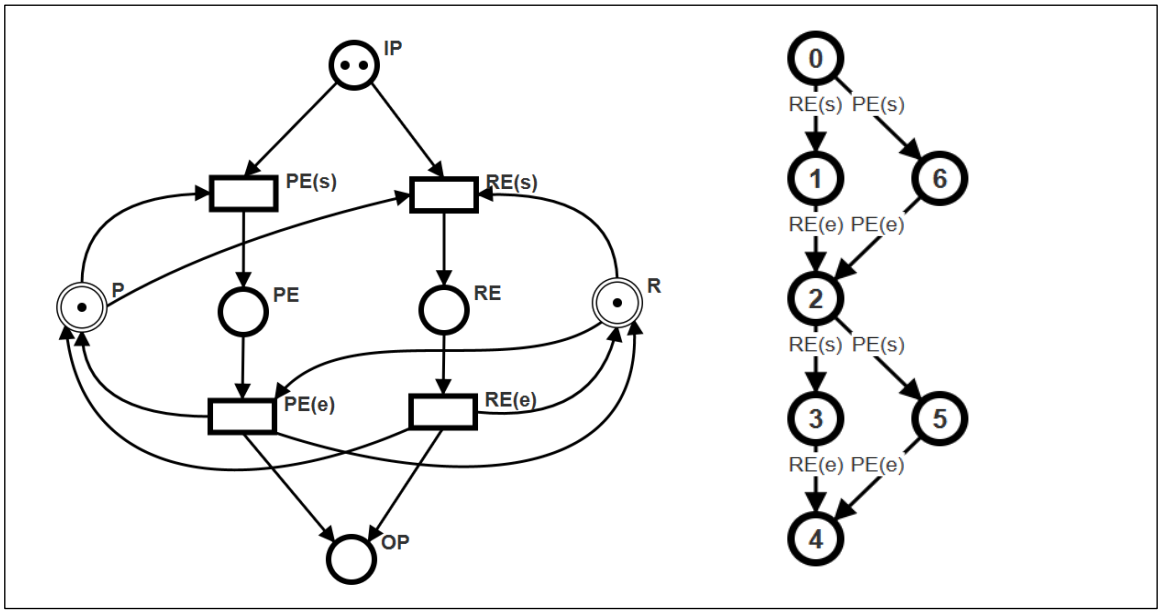
Nakonec proběhla simulace analyzované PPTN. Již v průběhu jednoho z prvních pokusů se PPTN dostala do stavu uváznutí (viz obr. 5-5).



Obrázek 5-5: PPTN ve stavu uváznutí, zdroj: vlastní

5.1.6 Vyhodnocení a úprava PPTN

Cílem této kapitoly je i poukázat na způsob, jak deadlock odstranit. Hrana, která původně vedla z místa *P* od přechodu *RE(e)* byla odstraněna. Místo ní byla definována hrana vedoucí z místa *P* do přechodu *RE(s)*. Na obrázku 5-6 je uveden nový návrh PPTN a její graf dosažitelných značení.



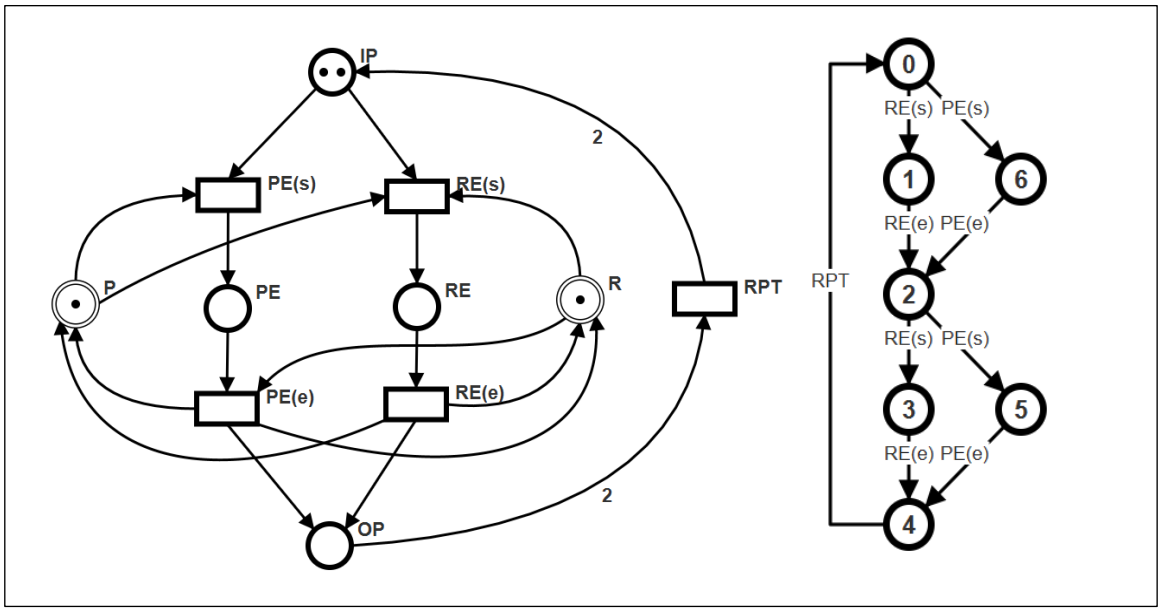
Obrázek 5-6: Upravená PPTN a její sekvenční diagram, zdroj: vlastní

Na základě grafu dosažitelných značení lze usoudit, že v této upravené PPTN ke stavu uváznutí již nedochází. PPTN se vždy ve výsledku dostane do výstupního značení M_4 (viz vzorec 5-3).

$$\mathrm{M}_4 = (0,2,1,0,1,0) \qquad \textit{Vzorec 5-3}$$

5.1.7 Zajištění žádaných vlastností

Žádaných vlastností lze v případě modelované PPTN dosáhnou přidáním přechodu *PRT* a dvou hran (*OP*, *RPT*) a (*RPT*, *IP*) (viz obr. 5-7).



Obrázek 5-7: Výsledný návrh PPTN s propojením výstupního a vstupního místa, zdroj: vlastní

Po této úpravě, je PPTN omezená, živá, slabě živá, silně propojená, reverzibilní a neobsahuje deadlock.

5.1.8 Vyhodnocení validace vyvinuté aplikace

Vyvinutá aplikace disponuje dostatečnými grafickými prostředky pro zpracování PPTN, základními analytickými prostředky (incidenční matice), na jejichž základě lze provádět další analýzu mimo aplikaci (např. analýzu p-invariantů). Z pohledu vlastností je možné nechat vygenerovat graf dosažitelných značení dané PPTN, na jehož základě lze vyhodnotit základní vlastnosti a v neposlední řadě i zjistit všechny stavy (značení) PPTN. Navrženou PPTN lze simulovat, a tím odhalit problémy daného návrhu, nebo naopak potvrdit, že návrh neobsahuje chyby.

5.2 Výrobní linka

Jako příklad z oblasti řízení výroby se nabízí modelování běhu výrobní linky. V praxi se na procesu výroby podílí více zdrojů (výrobní zařízení, zaměstnanci atd.). Tyto zdroje mezi sebou musí nějakým způsobem komunikovat a být navzájem korigovány, aby nedocházelo k prodlevám ve výrobě a ta mohla být kontinuální.

Vzorovým příkladem může být proces, kdy je produkce zařízení *A* využita zařízením *B* pro výrobu finálního produktu, který je poté balen do palet a odvážen

do skladu. Tento proces byl také vybrán pro validaci vyvinuté aplikace. Oproti předchozímu procesu (viz kapitola 5.1) hranová funkce *AF* sítě PPTN nabývá více hodnot (nejen hodnoty 1) a provádění přechodu je ovlivněno jejich prioritami.

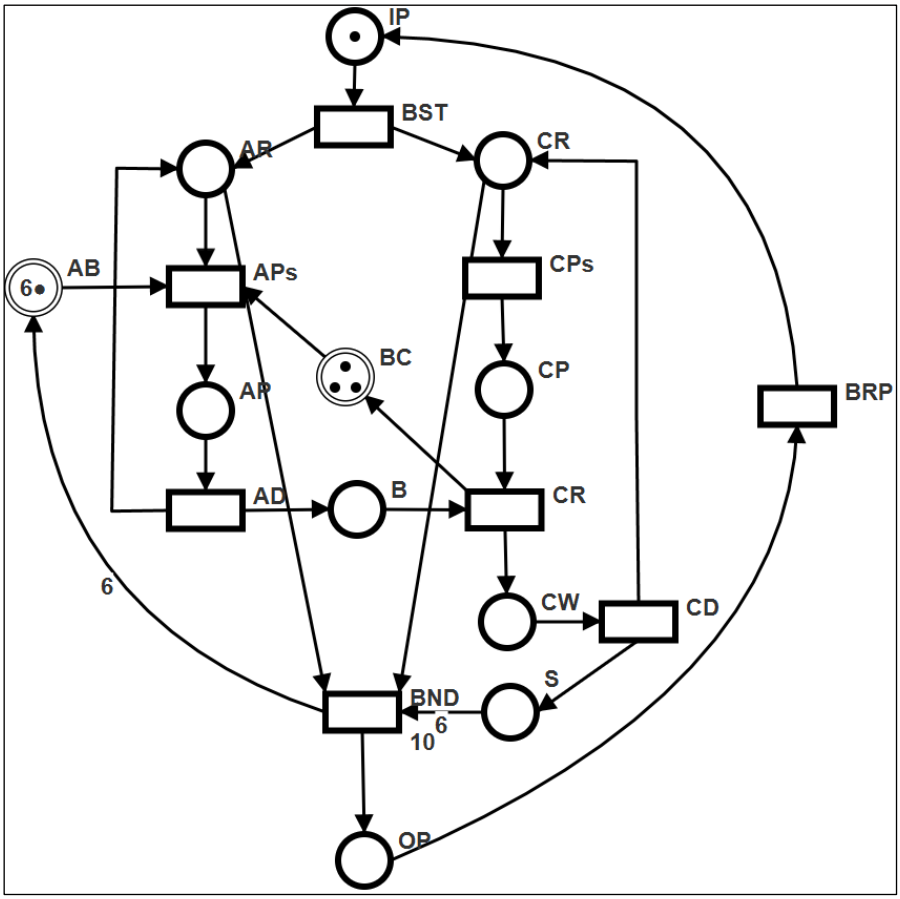
5.2.1 PPTN systému řízení

Výrobní zařízení *A* produkuje výrobky, které jsou zpracovávány výrobním zařízením *C*, a to tak, že jej zařízení *C* bere ze zásobníku *B*. Pro pochopení dalšího textu je doporučeno prostudovat tabulku 5-2 (slovo „*výrobní zařízení*“ bylo v tabulce vynecháno).

Tabulka 5-2: Komponenty PPTN, zdroj: vlastní

Zkratka	Kategorie	Název	Popis
IP	Vstupní m.	Input Place	Počátek procesu
AR	Místo	*A* ready	*A* je připraveno
BST	Přechod	Batch Start	Výroba dávky začala
CR	Místo	*C* ready	*C* je připraveno
AB	M. zdrojů	Batch size of *A*	Velikost dávky *A*
AP(s)	Přechod	Production of *A* started	Produkce *A* začala
CP(s)	Přechod	Preparation of *C* started	Příprava *C* začala
AP	Místo	*A* is prepared	*A* je připraveno
BC	M. zdrojů	Curr. free capacity of *B*	Současná volná kapacita *B*
CP	Přechod	*C* is ready	*C* je připraveno
BRP	Přechod	Batch Repeat Cycle	Cyklus opakování dávky
AD	Přechod	*A* is dispatching production to buffer	*A* expeduje produkci do zásobníku
B	Místo	Buffer	Zásobník
CR	Přechod	*C* is retrieving production from buffer	*C* odebírá produkci ze zásobníku
CW	Místo	*C* is working	*C* pracuje
CD	Přechod	*C* is dispatching production to stock	*C* expeduje produkci do skladu
BND	Přechod	Batch has been completed	Dávka byla zpracována
S	Místo	Stock	Sklad
OP	Výstupní m.	Output Place	Výstupní místo

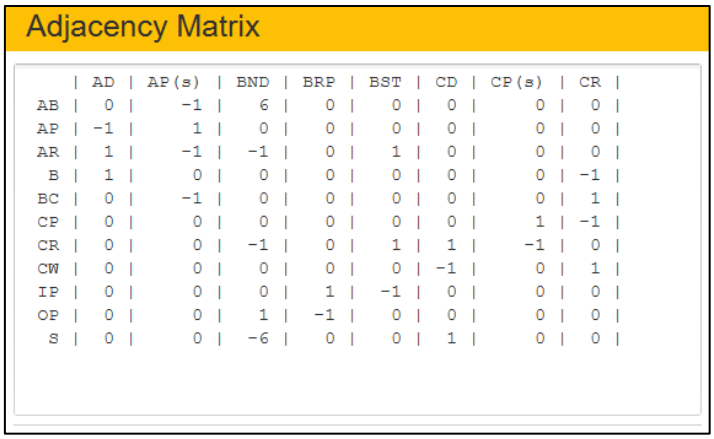
Výroba probíhá v cyklech, kdy v rámci jednoho cyklu dochází ke zpracování šesti (viz počet značek v místě zdrojů *AB*) surovin zařízením *A*. Zásobník *B* má kapacitu pouze pro 3 výrobky (viz počet značek v místě zdrojů *BC*) zařízení *A*. Pokud je zásobník plný, tedy v místě zdrojů *BC* se nenacházejí žádné značky, zařízení *A* čeká a neprodukuje výrobky. V okamžiku, kdy zařízení *C* vyrobilo 6 produktů (viz hodnota hranové funkce *AF* hrany (*S*, *BND*)), sklad je plný a musí dojít ke zpracování výroby, poté začíná celý cyklus znovu.



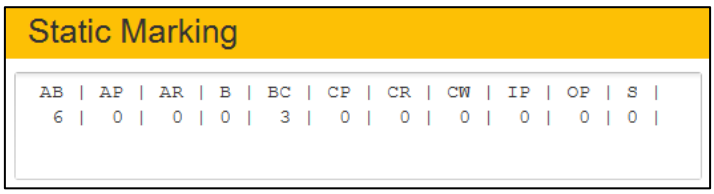
Obrázek 5-8: PPTN výrobní linky, zdroj: vlastní

5.2.2 Incidenční matice a značení PPTN

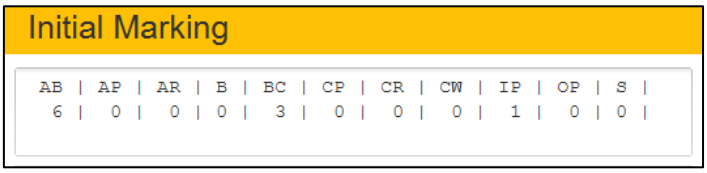
Incidenční matice PPTN, statické a vstupní (počáteční značení) je uvedeno na obrázcích 5-9, 5-10 a 5-11.



Obrázek 5-9: Incidenční matice, zdroj: vlastní



Obrázek 5-10: Statické značení PPTN, zdroj: vlastní



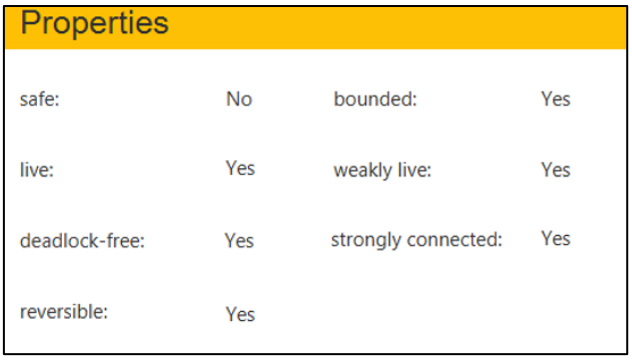
Obrázek 5-11: Vstupní značení PPTN, zdroj: vlastní

5.2.3 Graf dosažitelných značení

Na základě grafu dosažitelných značení (viz příloha č. 1) lze usoudit, že PPTN neobsahuje deadlock a nabývá 105 stavů, kdy minimální počet provedení přechodů pro uskutečnění jednoho výrobního cyklu je 32. Dále z grafu vyplývá, že je síť živá a reverzibilní, protože je výstupní značení M_{32} spojeno se vstupním značením M_0.

5.2.4 Základní vlastnosti

PPTN je dle dialogu základních vlastností (viz obr. 5-12) omezená, živá, slabě živá, silně propojená, reverzibilní a neobsahuje deadlock.



Obrázek 5-12: Dialog základních vlastností PPTN, zdroj: vlastní

5.2.5 Simulace

V rámci simulace bylo ověřeno, že za situace, kdy je proveditelný přechod *CP(s)* a *BND*, je proveden právě přechod *BND*, z důvodu, že je mu přiřazena vyšší priorita (10), a tento fakt lze ověřit i na základě GDZ. Výstupního značení je vždy dosaženo a provedením přechodu *BRP* se PPTN dostane do vstupního značení a proces se opakuje.

5.2.6 Vyhodnocení validace vyvinuté aplikace

Prostřednictvím této PPTN bylo úspěšně validováno, že vyvinutá aplikace disponuje pokročilými možnostmi podpory modelování PPTN. Priorita přechodů je tedy stejně jako hodnota hranové funkce *AF* brána v potaz při simulaci a analýze sítí. Pomocí aplikace lze tedy modelovat sítě o síle Turingova stroje.

5.3 Návrhy pro budoucí vývoj aplikace

Na základě validace aplikace pomocí dvou výše modelovaných procesů (viz kapitola 5.1 a 5.2) byly stanoveny návrhy pro budoucí vývoj aplikace, mezi které patří následující:

- **výpočet p-invariantů Petriho sítě** – v současné verzi aplikace je však možné vypočítat invarianty pomocí externí aplikace na základě vygenerované incidenční matice,
- **implementace podpory vysokoúrovňových Petriho sítí** – složitost PPTN roste v porovnání s vysokoúrovňovými Petriho sítěmi mnohem rychleji,
- **možnost dynamicky přesunovat popisky komponent Petriho sítě** – z hlediska UX však náročné případě mobilní verze aplikace.

6 Závěr

Cíle stanovené v úvodu této práce byly splněny.

Na základě uživatelského příběhu a teorie Petriho sítí byly definovány počáteční požadavky na aplikaci. Pro efektivní práci s aplikací byl navržen koncept workflow tvorby Petriho sítí, který zahrnuje fázi tvorby projektu, správy projektu a editace Petriho sítí. Ta se dále skládá ze čtyř procesů – grafický návrh, konfigurace a validace, simulace a analýza dat. Teorie Petriho sítí byla zdrojem výhradně funkčních požadavků z oblasti vlastností jednotlivých komponent, vlastností Petriho sítí a její validace.

Pomocí analýzy dostupných aplikací byly identifikovány technologie, na kterých je dostupný software postaven. Dále byly jednotlivé aplikace zhodnoceny z hlediska jejich uživatelského rozhraní, funkcionality a také potenciálního vývoje jejich verzí pro mobilní platformy. Autor práce nenalezl dostupné programové řešení pro práci s Petriho sítěmi pro desktopové a zároveň mobilní platformy v době výběru tématu a ani v době psaní této práce.

Multiplatformní aplikace pro podporu modelování a simulace Petriho sítí byla úspěšně navržena a je implementována. Jsou podporovány čtyři cílové platformy (kromě iOS) a vznikly dvě verze aplikace – pro desktopové a mobilní platformy. Pro mobilní platformy je aplikace navíc optimalizována tak, aby bylo zajištěno její bezproblémové využívání na menších zobrazovacích zařízeních, než jsou osmipalcové tablety.

Verzi aplikace pro mobilní zařízení bylo možné vyvinout díky projektu Gluon Mobile. Jedním z cílů této práce bylo ověřit, zda jej lze využít v praxi, a pokud ano, tak v jakém rozsahu. Vyvíjená aplikace je úspěšně uzpůsobena pro mobilní platformy s tím, že bylo nutné vyvarovat se použití některých novějších prvků programovacího jazyka Java a aplikaci se nezdařilo kvůli její náročnosti zprovoznit na platformě iOS. Pro řešení problému byla však stanovena východiska pro budoucí vývoj aplikace (viz kapitola 4.5.5). Co se týče výkonu – jediným problémem jsou méně plynulé animace a efekty na méně výkonných zařízeních, což by mělo v rámci budoucího vývoje projektu Gluon Mobile řešeno, protože pro běh aplikací vyvíjených nativně pro danou platformu není potřeba tak vysoký grafický výkon zařízení.

Vyvíjenou aplikaci bylo nutné ve výsledku validovat na jednoduchém příkladu z praxe. Ukázalo se, že pomocí aplikace lze modelovat, konfigurovat, validovat a simulovat Procesní P/T Petriho sítě a zjišťovat jejich základní vlastnosti.

Koncept a vyvíjená aplikace byla poprvé představena v rámci příspěvku *MPNP: Petri Net Processes Modeling and Simulation Tool for Mobile Devices* autorů Čechák a Martiník (2016) na konferenci *Information Technology for Practice 2016*.

Seznam použité literatury

Odborné knihy

AALST, Wil van der. *Timed coloured Petri nets and their application to logistics*. Eindhoven, 1992. Disertační práce. Eindhoven University of Technology.

ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací.* Brno, Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.

DICESARE, F., G. HARHALAKIS, J. M. PROTH, M. SILVA, F. VERNADAT. *Practice of Petri Nets in Manufacturing.* Berlin: Springer-Verlag, 1993. ISBN 978-94-011-6957-8.

DIETZ, Jan L. G. *Enterprise ontology: theory and methodology*. New York: Springer, 2006. ISBN 978-3-540-29169-5.

FIALA, Josef a Jan MINISTR. *Průvodce analýzou a modelováním procesů.* Ostrava: VŠB-TU Ostrava, 2003. 110 s. ISBN 20-248-0500-6.

HAMILTON, Kim and Russell MILES. *Learning UML 2.0.* Sebastopol: O'Reilly Media, 2006. 286 p. ISBN 978-0-59-600982-3.

HRUBÝ, Pavel. *Víceúrovňové modelování podnikových procesů (systém REA).* Ostrava: VŠB-TU, 2010. ISBN 978-80-248-2334-8.

HUANG, Hejiao. *Property-preserving petri net process algebra: in software engineering*. Hackensack, N.J.: World Scientific, c2012. ISBN 978-981-4324-28-1.

MARTINÍK, Ivo. *Modelování objektově orientovaných programových systémů Petriho sítěmi*. Ostrava: VŠB-TU Ostrava, 2015. ISBN 978-80-248-3807-6.

REISIG, Wolfgang. *Understanding Petri Nets: modeling techniques, analysis methods, case studies*. Berlin: Springer, 2013. ISBN 978-3-642-33277-7.

SALATINO, Mauricio and Esteban ALIVERTI. *jBPM5 Developer Guide.* Birmingham: Packt Publishing, 2012. 340 p. ISBN 978-1-84951-644-0.

SHARAN, Kishori. *Learn JavaFX 8: Building User Experience and Interfaces with Java 8*. New York: Apress, 2015. ISBN 978-1-484211-43-4.

SMITH, Dave. *Android recipes: a problem-solution approach for Android 5.0*. Fourth ed., New York: Apress, 2015. ISBN 978-1-484204-76-4.

ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. Brno: Computer Press, 2014. ISBN 978-80-251-4194-6.

VLČEK, Pavel a Ludmila KALUŽOVÁ. *Základy algoritmizace a Microsoft Access*. Ostrava: JOKL, 2012. ISBN 978-80-260-1592-5.

Články z konferencí

ČECHÁK, Martin and Ivo MARTINÍK. *MPNP: Petri Net Processes Modeling and Simulation Tool for Mobile Devices. In Proceedings of the 19th International Conference on Information Technology for Practice 2016*, Faculty of Economics,VŠB - Technical University of Ostrava, 13. 10. - 14. 10. 2016. Ostrava: VŠB – Technical University of Ostrava, 2016, s. 111-118. ISBN 978-80-248-3970-7.

SHEN, L., Q. CHEN a J.Y.S. LUH. *Truncation of Petri net models for simplifying computation of optimum scheduling problems. Computers in Industry* [online]. 1992, 20(1), 25-43 [cit. 2017-04-16]. DOI: 10.1016/0166-3615(92)90125-7. ISSN 0166-3615. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0166361592901257

JEETENDRA, V. A., O. V. KRISHNAIAH CHETTY a J. PRASHANTH REDDY*. Petri Nets for Project Management and Resource Levelling. The International Journal of Advanced Manufacturing Technology* [online]. 2000-6-8, 16(7), 516-520 [cit. 2017-04-16]. DOI: 10.1007/s001700070059. ISSN 0268-3768. Dostupné z: http://link.springer.com/10.1007/s001700070059

MARTINÍK, Ivo. *PLACE_SUBST Transformation of P/T Petri Process Nets and Its Properties. Computer Information Systems and Industrial Management*. Lecture Notes in Computer Science. Volume 9842. 2016, s. 504-515. ISSN 0302-9743.

VAN HEE, Kees M., Natalia SIDOROVA a Jan Martijn VAN DER WERF. Business Process Modeling Using Petri Nets [online]. s. 116 [cit. 2017-04-12]. 2013. DOI: 10.1007/978-3-642-38143-0_4. Dostupné z: http://link.springer.com/10.1007/978-3-642-38143-0_4

POLIASCHUK, N. *Simulation of Economic Costs Systems Using Petri Nets*. 2011. In: Anals of Marketing-mba. Volume 4 (2). 1-17.

Elektronické dokumenty a ostatní

Android Developers. *Dashboards* [online]. c2017 [cit. 2017-04-14]. Dostupné z: https://developer.android.com/about/dashboards/index.html

Apple Developer, *App Store* [online]. 2017 [cit. 2017-04-04]. Dostupné z: https://developer.apple.com/support/app-store/

BECKWITH, Monica. *Ahead-of-Time (AOT) Compilation May Come to OpenJDK HotSpot in Java 9*. Infoq [online]. 1. 9. 2016 [cit. 2017-04-08]. Dostupné z: https://www.infoq.com/news/2016/10/AOT-HotSpot-OpenJDK-9

CPN Tools. *CPN Tools* [online]. c2017 [cit. 2017-04-08]. Dostupné z: http://cpntools.org/

ČECHÁK, Martin, *Návrh a implementace IS pro cateringovou společnost.* Ostrava, 2015. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava. Ekonomická fakulta.

GIBBS, Samuel. *From Windows 1 to Windows 10: 29 years of Windows evolution*. Theguardian [online]. 2. 9. 2014 [cit. 2017-04-08]. Dostupné z: https://www.theguardian.com/technology/2014/oct/02/from-windows-1-to-windows-10-29-years-of-windows-evolution

Gluon. *Rapid Enterprise Mobile Apps: Build, Connect, Manage with Gluon* [online]. c2017a [cit. 04.04.2017]. Dostupné z: http://gluonhq.com/products/mobile/

Gluon. *Gluon VM: Superior Java 9 performance on iOS & Android* [online]. c2017b [cit. 2017-04-17]. Dostupné z: http://gluonhq.com/products/mobile/vm/

DOBIE, A., R. HOLLY, J. HILDENBRAND, A. MARTONIK. *Android Pre-History*. Android Central [online]. c2016 [cit. 2017-04-08]. Dostupné z: http://www.androidcentral.com/android-pre-history

Dokumentace ORACLE. [online] c2008, 2015 [cit. 2017-04-08]. Dostupné z: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Pane.html

Gradle Inc. *Chapter 1. Introduction* [online]. c2017 [cit. 2017-04-04]. Dostupné z: https://docs.gradle.org/3.4.1/userguide/introduction.html#gsc.tab=0

GREBEŇ, David. *Kompletní historie OS X: Od první verze až po El Capitan*. Letem Světem Applem [online]. 2016 [cit. 2017-04-08]. Dostupné z: https://www.letemsvetemapplem.eu/2016/03/13/kompletni-historie-os-x/

IDC. *Smartphone OS Market Share, 2016 Q3* [online]. c2017 [cit. 2017-04-10]. Dostupné z: http://www.idc.com/promo/smartphone-market-share

JANIŠ, Petr. *Jak využít Kanban při vývoji software* [online]. 23. 9. 2013 [cit. 2017-04-04]. Dostupné z: http://www.projectman.cz/clanky/posts/35-jak-vyuzit-kanban-pri-vyvoji-software

RADIGAN, Dan. ATLASSIAN. *Kanban: How the kanban methodology applies to software development* [online]. c2017 [cit. 2017-04-04]. Dostupné z: https://www.atlassian.com/agile/kanban

Root.cz. *Základy Linuxu* [online]. c1998-2017 [cit. 2017-30-06]. Dostupné z: https://www.root.cz/texty/zaklady-linuxu/

Smartsheet. *The Newbie's Complete Guide to Kanban by Top Experts* [online]. c2017 [cit. 2017-03-20]. Dostupné z: https://www.smartsheet.com/complete-kanban-project-management-guide-newbies-top-pm-experts

TAPAAL. *TAPAAL: Tool for Verification of Timed-Arc Petri Nets* [online]. c2008-2016 [cit. 2017-03-20]. Dostupné z: http://www.tapaal.net/

ZYL, Jason van. *History of Maven* [online]. 2017 [cit. 2017-04-04]. Dostupné z: https://maven.apache.org/background/history-of-maven.html

Seznam zkratek

AF	Arch Function
AOT	Ahead-of-Time
API	Appplication Programming Interface
ARM	Advanced RISC Machine
BOG	Bipartite Oriented Graph
BPM	Business Process Management
BPMN	Business Process Modelling Notation
BSD	Berkley Software Distribution
BSON	Binary JSON
CAD	Computer Aided Design
CSV	Comma Separated Values
DAO	Data Access Object
DB	Database
DEMO	Design and Engineering Methodology for Organizations
DI	Dependency Injection
FXML	JavaFX XML
GDZ	Graf dosažitelných značení
GNU	GNU General Public Licence
GPL	General Public Licence
GPS	Global Positioning Systém
GUI	Graphical User Interface
HMVC	Hierarchical MVC
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

IP	Input Place
Java ME	Java Micro Edition
JDK	Java Development Kit
JIT	Just-in-Time
JRE	Java Runtime Edition
JSON	Javascript Object Notation
JVM	Java Virtual Machine
LGPL	Lesser General Public Licence
MVC	Model-View-Controller
NET	Síť
OG	orientovaný graf
OMG	Object Management Group
OOP	Object Oriented Programming
OP	Output Place
OS	Operation System
PN	Petri Net
PNG	Portable Network Graphics
POM	Project Object Model
PPPA	Property-Preserving Petri Net Process Algebra
PPTN	Process P/T Petri Net
PTN	P/T Petri Net
REA	Resource Event Agent
RISC	Reduced Instruction Set Computing
RP	konečná množina míst zdrojů

RUP	Rational Unified Process
SD	Secure Digital
TP	Transition Priority
UI	User Interface
UML	Unified Modelling Language
UP	Unified Process
UX	User Experience
XML	eXtensible Markup Language
XP	Extreme Programming

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že:

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Karviné dne 19. 4. 2017

Čechák

.....

Bc. Martin Čechák

Seznam příloh

Příloha č. 1: Graf dosažitelných značení PPTN výrobní linky

Příloha č. 2: Přiložené DVD s následujícími soubory:

- spustitelným souborem typu jar (desktopová verze aplikace).
- instalačním souborem typu apk (experimentální mobilní verze pro OS Android 4.4 a vyšší, pouze pro architekturu ARM),
- textový soubor s licencemi využitých ikon.

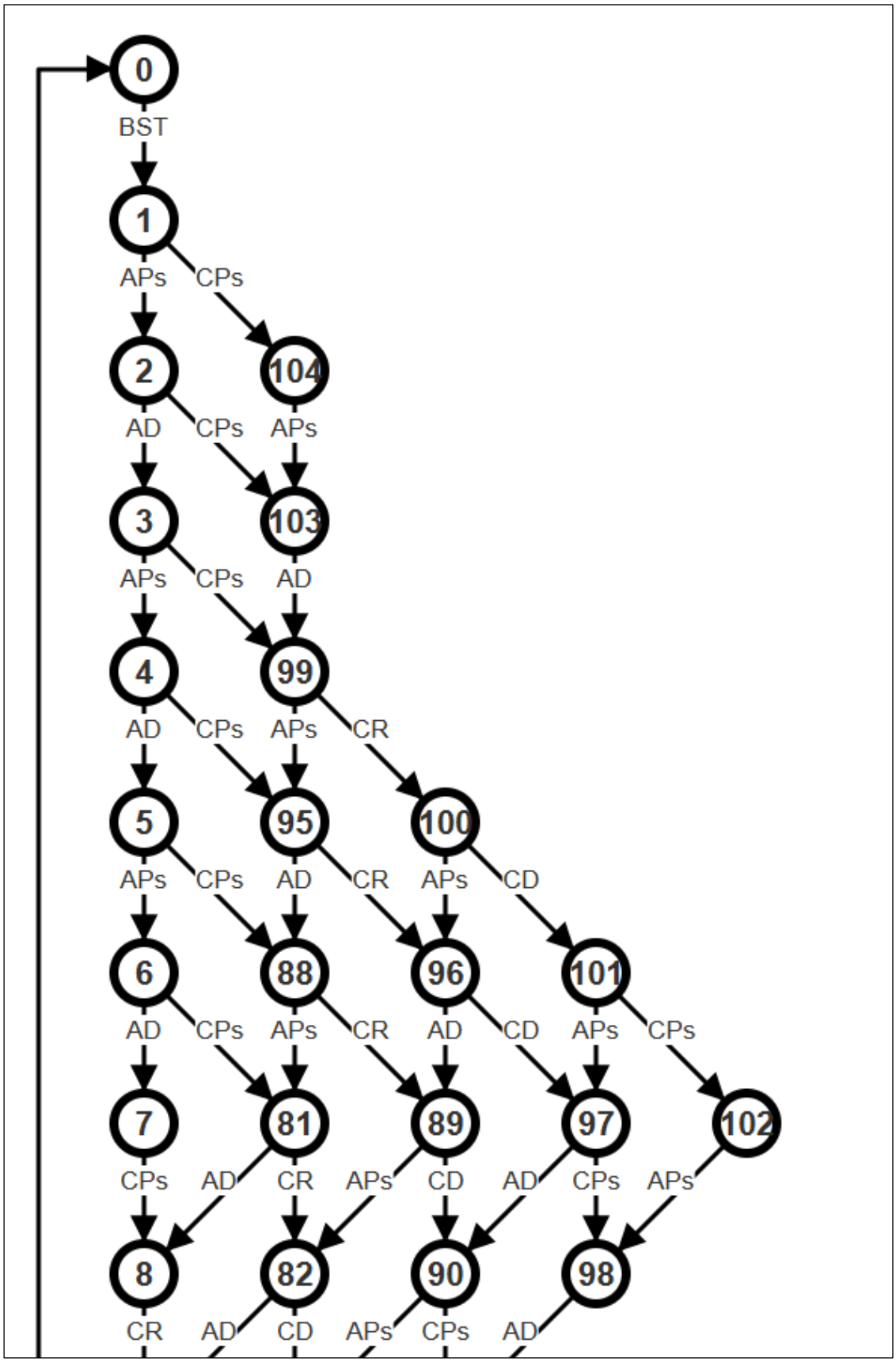
Příloha č. 1: Graf dosažitelných značení PPTN výrobní linky

Ve vzorci 1 je uvedeno pořadí počtů značek jednotlivých v jednotlivých míst, což platí pro obrázek 1. Na obrázcích č. 2, 3, 4 a 5 je uveden GDZ sítě PPTN výrobní linky.

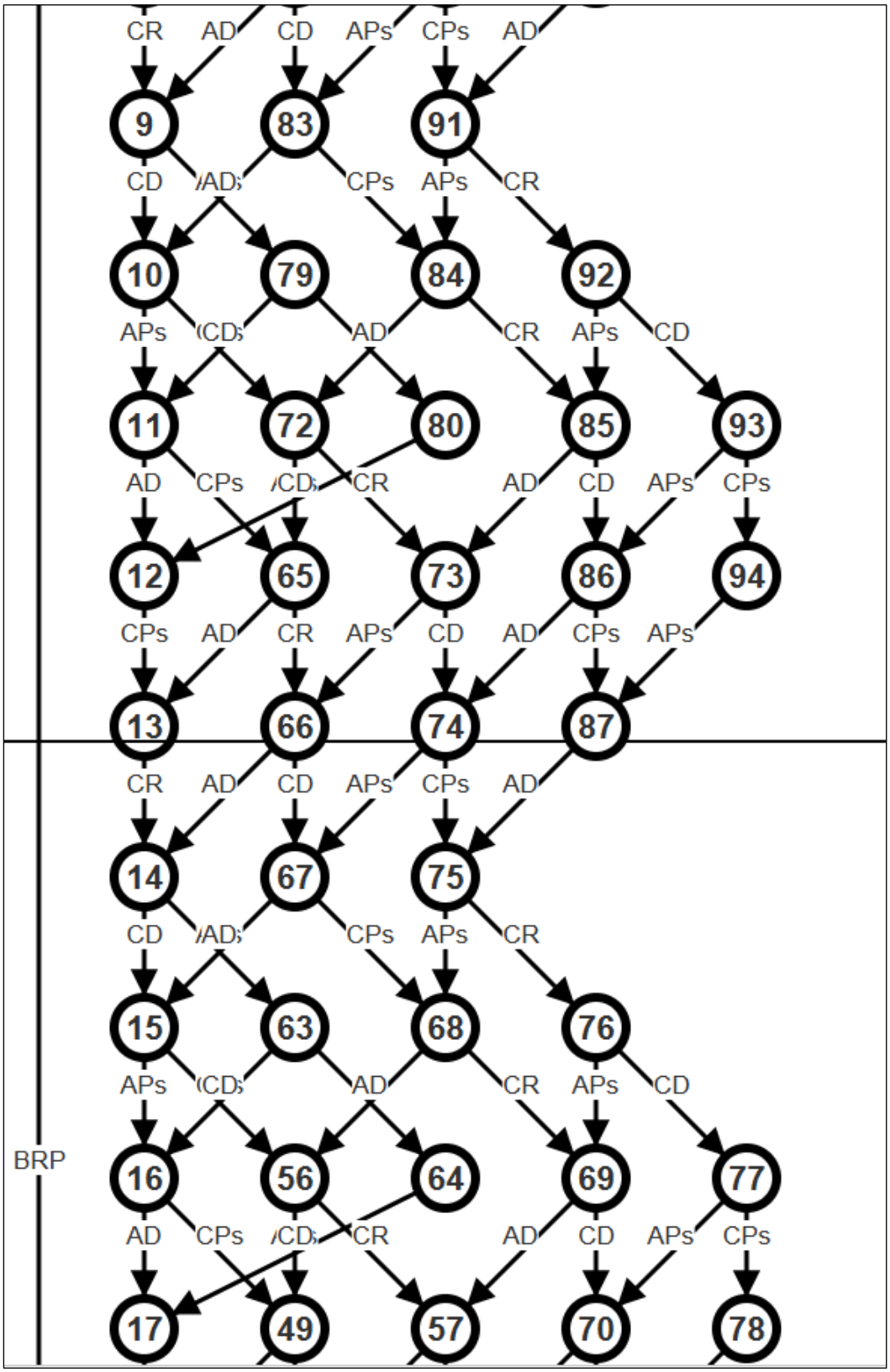
$$\mathrm{M}_x = (|AB|, |AP|, |AR|, |B|, |BC|, |CP|, |CR|, |CW|, |IP|, |OP|, |S|) \qquad \textit{Vzorec 1}$$

0	(6,0,0,0,3,0,0,0,1,0,0)
1	(6,0,1,0,3,0,1,0,0,0,0)
2	(5,1,0,0,2,0,1,0,0,0,0)
3	(5,0,1,1,2,0,1,0,0,0,0)
4	(4,1,0,1,1,0,1,0,0,0,0)
5	(4,0,1,2,1,0,1,0,0,0,0)
6	(3,1,0,2,0,0,1,0,0,0,0)
7	(3,0,1,3,0,0,1,0,0,0,0)
8	(3,0,1,3,0,1,0,0,0,0,0)
9	(3,0,1,2,1,0,0,1,0,0,0)
10	(3,0,1,2,1,0,1,0,0,0,1)
11	(2,1,0,2,0,0,1,0,0,0,1)
12	(2,0,1,3,0,0,1,0,0,0,1)
13	(2,0,1,3,0,1,0,0,0,0,1)
14	(2,0,1,2,1,0,0,1,0,0,1)
15	(2,0,1,2,1,0,1,0,0,0,2)
16	(1,1,0,2,0,0,1,0,0,0,2)
17	(1,0,1,3,0,0,1,0,0,0,2)
18	(1,0,1,3,0,1,0,0,0,0,2)
19	(1,0,1,2,1,0,0,1,0,0,2)
20	(1,0,1,2,1,0,1,0,0,0,3)
21	(0,1,0,2,0,0,1,0,0,0,3)
22	(0,0,1,3,0,0,1,0,0,0,3)
23	(0,0,1,3,0,1,0,0,0,0,3)
24	(0,0,1,2,1,0,0,1,0,0,3)
25	(0,0,1,2,1,0,1,0,0,0,4)
26	(0,0,1,2,1,1,0,0,0,0,4)
27	(0,0,1,1,2,0,0,1,0,0,4)
28	(0,0,1,1,2,0,1,0,0,0,5)
29	(0,0,1,1,2,1,0,0,0,0,5)
30	(0,0,1,0,3,0,0,1,0,0,5)
31	(0,0,1,0,3,0,1,0,0,0,6)
32	(6,0,0,0,3,0,0,0,0,1,0)
33	(0,1,0,2,0,1,0,0,0,0,3)
34	(0,1,0,1,1,0,0,1,0,0,3)
35	(0,1,0,1,1,0,1,0,0,0,4)
36	(0,1,0,1,1,1,0,0,0,0,4)
37	(0,1,0,0,2,0,0,1,0,0,4)
38	(0,1,0,0,2,0,1,0,0,0,5)
39	(0,1,0,0,2,1,0,0,0,0,5)
40	(1,0,1,2,1,1,0,0,0,0,3)
41	(1,0,1,1,2,0,0,1,0,0,3)
42	(1,0,1,1,2,0,1,0,0,0,4)
43	(1,0,1,1,2,1,0,0,0,0,4)
44	(1,0,1,0,3,0,0,1,0,0,4)
45	(1,0,1,0,3,0,1,0,0,0,5)
46	(1,0,1,0,3,1,0,0,0,0,5)
47	(0,1,0,2,0,0,0,1,0,0,2)
48	(0,0,1,3,0,0,0,1,0,0,2)
49	(1,1,0,2,0,1,0,0,0,0,2)
50	(1,1,0,1,1,0,0,1,0,0,2)
51	(1,1,0,1,1,0,1,0,0,0,3)
52	(1,1,0,1,1,1,0,0,0,0,3)
53	(1,1,0,0,2,0,0,1,0,0,3)
54	(1,1,0,0,2,0,1,0,0,0,4)
55	(1,1,0,0,2,1,0,0,0,0,4)
56	(2,0,1,2,1,1,0,0,0,0,2)
57	(2,0,1,1,2,0,0,1,0,0,2)
58	(2,0,1,1,2,0,1,0,0,0,3)
59	(2,0,1,1,2,1,0,0,0,0,3)
60	(2,0,1,0,3,0,0,1,0,0,3)
61	(2,0,1,0,3,0,1,0,0,0,4)
62	(2,0,1,0,3,1,0,0,0,0,4)
63	(1,1,0,2,0,0,0,1,0,0,1)
64	(1,0,1,3,0,0,0,1,0,0,1)
65	(2,1,0,2,0,1,0,0,0,0,1)
66	(2,1,0,1,1,0,0,1,0,0,1)
67	(2,1,0,1,1,0,1,0,0,0,2)
68	(2,1,0,1,1,1,0,0,0,0,2)
69	(2,1,0,0,2,0,0,1,0,0,2)
70	(2,1,0,0,2,0,1,0,0,0,3)
71	(2,1,0,0,2,1,0,0,0,0,3)
72	(3,0,1,2,1,1,0,0,0,0,1)
73	(3,0,1,1,2,0,0,1,0,0,1)
74	(3,0,1,1,2,0,1,0,0,0,2)
75	(3,0,1,1,2,1,0,0,0,0,2)
76	(3,0,1,0,3,0,0,1,0,0,2)
77	(3,0,1,0,3,0,1,0,0,0,3)
78	(3,0,1,0,3,1,0,0,0,0,3)
79	(2,1,0,2,0,0,0,1,0,0,0)
80	(2,0,1,3,0,0,0,1,0,0,0)
81	(3,1,0,2,0,1,0,0,0,0,0)
82	(3,1,0,1,1,0,0,1,0,0,0)
83	(3,1,0,1,1,0,1,0,0,0,1)
84	(3,1,0,1,1,1,0,0,0,0,1)
85	(3,1,0,0,2,0,0,1,0,0,1)
86	(3,1,0,0,2,0,1,0,0,0,2)
87	(3,1,0,0,2,1,0,0,0,0,2)
88	(4,0,1,2,1,1,0,0,0,0,0)
89	(4,0,1,1,2,0,0,1,0,0,0)
90	(4,0,1,1,2,0,1,0,0,0,1)
91	(4,0,1,1,2,1,0,0,0,0,1)
92	(4,0,1,0,3,0,0,1,0,0,1)
93	(4,0,1,0,3,0,1,0,0,0,2)
94	(4,0,1,0,3,1,0,0,0,0,2)
95	(4,1,0,1,1,1,0,0,0,0,0)
96	(4,1,0,0,2,0,0,1,0,0,0)
97	(4,1,0,0,2,0,1,0,0,0,1)
98	(4,1,0,0,2,1,0,0,0,0,1)
99	(5,0,1,1,2,1,0,0,0,0,0)
100	(5,0,1,0,3,0,0,1,0,0,0)
101	(5,0,1,0,3,0,1,0,0,0,1)
102	(5,0,1,0,3,1,0,0,0,0,1)
103	(5,1,0,0,2,1,0,0,0,0,0)
104	(6,0,1,0,3,1,0,0,0,0,0)

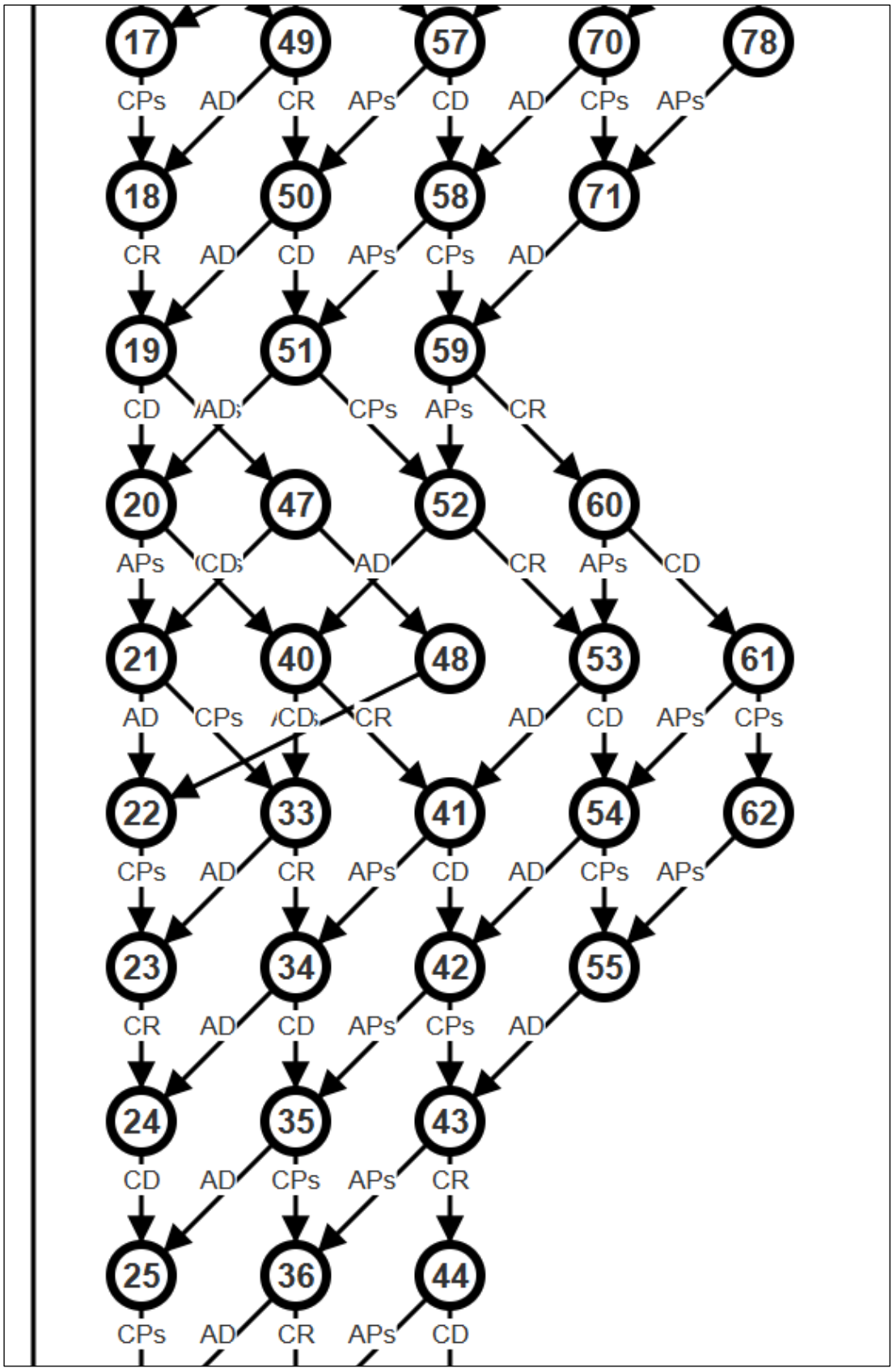
Obrázek 1: Seznam značení, zdroj: vlastní



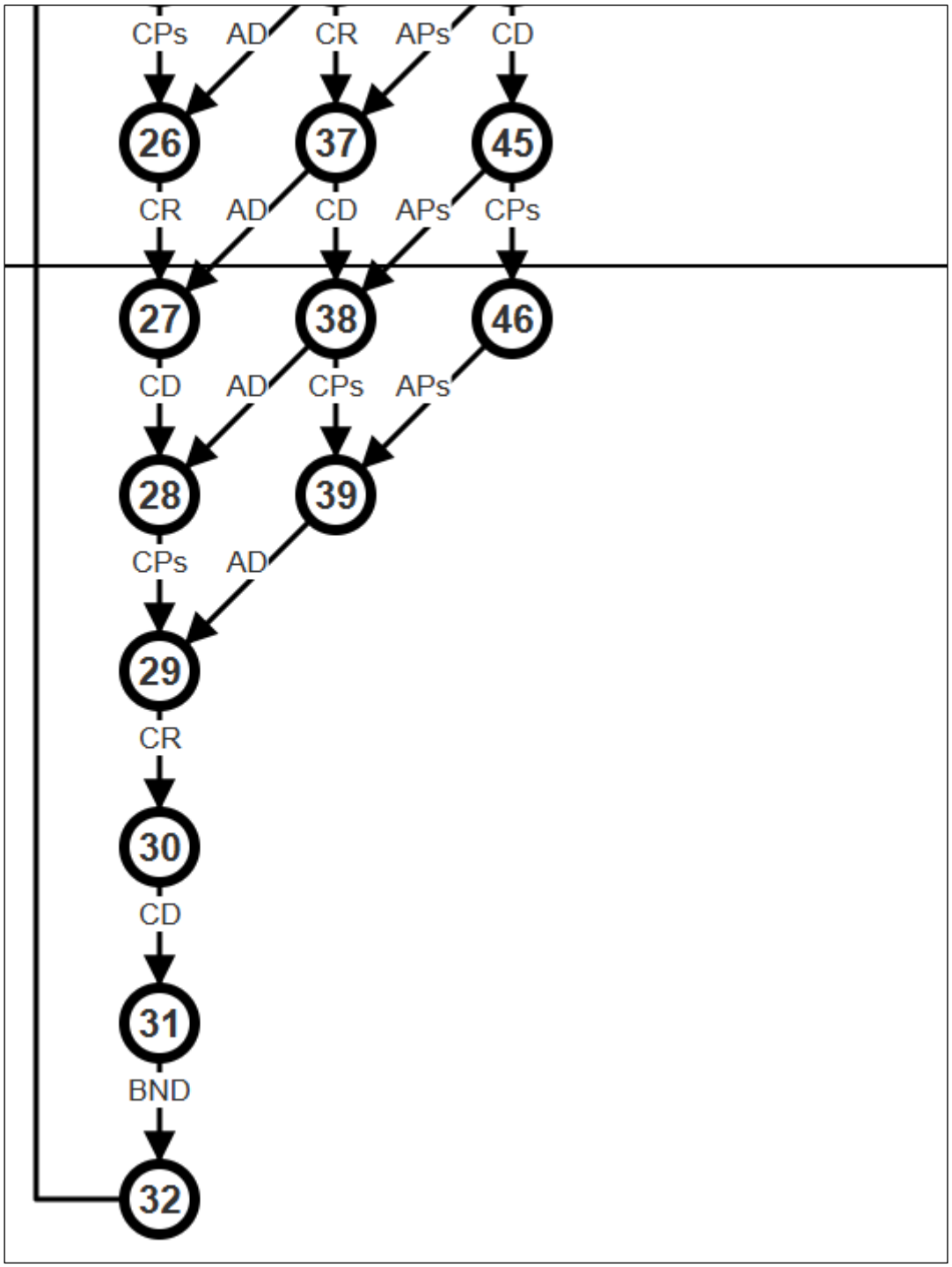
Obrázek 2: GDZ část 1., zdroj: vlastní



Obrázek 3: GDZ část 2., zdroj: vlastní



Obrázek 4: GDZ část 3., zdroj: vlastní



Obrázek 5: GDZ část 4., zdroj: vlastní